Grado en Ingeniería Informática
Especialidad de Computación

Aprendizaje por Refuerzo Aplicado a Personajes No Controlables en Minetest

Autor:
Aitor Romero Reviriego

Director:
Javier Béjar Alonso

Enero 2019

Resumen

La inteligencia artificial avanza con pasos de gigante una gran variedad de campos que jamás esperaríamos, pero hay uno de ellos con el cual tiene una larga historia y jamás abandonará: el mundo de los videojuegos. Este campo supone un lugar extraordinario de entrenamiento y lleno de retos para la IA donde a día de hoy ha conquistado muchos juegos de tipos diferentes y ha conseguido derrotar a profesionales de los mismos. En la actualidad, los juegos de tipo voxel como Minetest son bastante populares ya que ofrecen a los jugadores infinitas posibilidades de como jugar. Este proyecto intenta desarrollar agentes de aprendizaje por refuerzo para el videojuego Minetest que actúen como aliados del jugador. Para ello se diseñaran escenarios concretos con tareas comunes que podrían darse en cualquier momento en una partida del juego y se sacarán conclusiones.

Resum

La intel·ligència artificial avança amb passos de gegant entre una gran varietat de camps que mai esperaríem, però n'hi ha un d'ells amb el qual té una llarga historia y mai abandonarà: el mon dels videojocs. Aquest camp es un lloc extraordinari d'entrenament y ple de reptes per la IA on a dia d'avui ha conquistat molts jocs de tipus diferents y ha aconseguit vèncer a professionals dels mateixos. A la actualitat, els jocs de tipus voxel com Minetest son bastant populars degut a que ofereixen als jugador infinites possibilitats de com jugar. Aquest projecte intenta desenvolupar agents de aprenentatge per reforç per al videojoc Minetest que actuïn com aliats del jugador. Per això es dissenyaran escenaris concrets amb tasques comuns que podrien donar-se en qualsevol moment en una partida del joc y es trauran conclusions.

Abstract

Artificial intelligence is progressing at a fast pace in a wide variety of fields that we would never have expected. However, there is one with a long history of artificial intelligence that just keeps on growing: the world of videogames. This field is the perfect place for training and it is full of challenges. IA has conquered a lot of different games, even achieved to defeat professional players. Nowadays, voxel type games like Minetest are so popular due to the unlimited possibilities they offer when it comes to how to play the game. This project tries to develop reinforce learning agents in the Minetest videogame that act like player's allies. To accomplish our goal we will design specific stages with common gameplay events that would occur in any standard playthrough, and with this knowledge we will be able to reach our conclusions.

Agradecimientos

En este apartado me gustaría darle importancia a todas aquellas personas que sin ellos esta carrera no hubiera sido lo mismo. Gracias a:

Mi director de TFG y profesor de IA, CAIM y APA, Javier Béjar, por todas las reuniones y horas que le ha dedicado a este proyecto y por sus consejos que han ayudado a que haya finalizado en la forma que está ahora. Además, no quiero destacar solo su trabajo sino también las partes en las reuniones que hablábamos sobre IA, que han sido realmente interesantes, o cuando reíamos por algún comportamiento absurdo que había desarrollado algún agente o jugador al principio del entrenamiento o la implementación.

Mi madre y mi padre, Ana y Rafa, por haber estado a mi lado todo este tiempo que ha durado la carrera. Que como ellos dicen cuando hay un examen: "Tú estás nervioso, pero nosotros lo estamos aún más y eso que no nos vamos a examinar". Su apoyo ha sido enorme y han trabajado duro muchos años para permitirme elegir la carrera que quisiera y que la pudiera realizar.

A mi familia: titos, titas, primos y hermana. Es verdad que no nos vemos diariamente todos como si puedo ver a mis padres y mi hermana, pero seguro que estaréis de acuerdo que somos una familia un poco "loca", sin embargo, es en el buen sentido de la palabra. Cuando nos juntamos todos nos lo pasamos de coña y, queramos o no, en esas horas que pasamos juntos desconectamos de las responsabilidades que tenemos cada uno y estar juntos nos hace coger fuerza para seguir dando batalla días, semanas y meses más.

A mis amigos, por todos los buenos momentos que hemos pasado durante esta época de mi vida y los quedan por llegar. Hay a muchas personas y compañeros que me gustaría nombrar y que me han ayudado mucho durante estos años, pero me gustaría destacar a aquellas personas que han sido muy importantes para mí: a Erik, que lo conocí en 1ro de la ESO y sigue siendo uno de mis mejores amigos a día de hoy. Cada uno optó por seguir un camino diferente e intentamos esforzarnos y hacerlo lo mejor posible día a día. Como seguimos en contacto podemos ver claramente ese esfuerzo y eso hace que nos demos fuerzas el uno al otro. A Carlos, que le conocí a principio de carrera en la universidad y desde entonces no hemos hecho otra cosa que intentar coincidir en asignaturas y superar las dificultades que nos ofrecían juntos. Además, compartimos muchas aficiones y siempre encontramos una buenas razones para quedar entre las cuales se encuentran: "tengo que comprar lentillas, ha salido un juego nuevo y habrá que probarlo" entre otras. A David, otra de las personas que conocí a principio de

carrera y hemos pasado mucho tiempo juntos hablando de todo. Nos encantan los mismos tipos de videojuegos y siempre es divertido hablar con él. Últimamente hemos perdido el contacto, pero sé que tarde o temprano lo acabaremos recuperando. Sobre todo, destacar su puntualidad. A Alberto, persona que he conocido a final de la carrera y me ha apoyado desde entonces. Fui su mentor en el programa de mentorías de la FIB, igual que el de muchos otros, pero con él realmente hubo una conexión que no sucedió con los demás. Es como si tuviera en la carrera a un hermano menor. Le vi entrar, le seguiré apoyando y le veré acabar.

¡Gracias a todos!

Índice

1. Contexto y Alcance 1

1.1. Contexto 1

1.1.1. Introducción 1

1.1.2. Formulación del problema 2

1.1.3 Actores implicados 3

1.2. Estado del Arte 4

1.3. Objetivos 7

1.4. Alcance 8

1.5. Posibles Obstáculos 8

1.6. Metodología y Rigor 9

1.7. Análisis de la Metodología y Rigor 10

1.8. Integración de Conocimientos 10

2. Planificación Temporal 12

2.1. Programa 12

2.2. Descripción de las Tareas 12

2.2.1. Aprender sobre el aprendizaje por refuerzo 12

2.2.2. Instalación y configuración del entorno 13

2.2.3. Familiarización con Minetest y su conexión con JetBrains PyCharm Edu... 13

2.2.4. Diseño y selección de los escenarios de entrenamiento 14

2.2.5. Implementación, entrenamiento, evaluación del rendimiento de los agentes y prueba final 15

2.2.6. Análisis y conclusiones 15

2.2.7. Memoria del proyecto 15

2.2.8. Preparación de la presentación final del proyecto 16

2.3. Plan de acción y valoración de alternativas 16

2.4. Análisis actual de las alternativas 16

2.5. Recursos **17**
2.6. Estimación de Horas y Dependencias Inicial **18**
2.7. Estimación de Horas y Dependencias Final **19**
2.7.1. Cambios respecto la planificación inicial 20
2.8. Diagrama de Gantt Inicial del Proyecto **21**
2.9. Diagrama de Gantt Definitivo del Proyecto **22**
3. Coste y sostenibilidad **23**
3.1. Gestión Económica **23**
3.1.1. Consideraciones Iniciales 23
3.1.2. Presupuesto de Recursos Humanos Inicial 23
3.1.3. Presupuesto de Hardware 24
3.1.4. Presupuesto Software 25
3.1.5. Costes Indirectos 25
3.1.6. Imprevistos 25
3.1.7. Presupuesto Total Inicial 26
3.1.8. Presupuesto de Recursos Humanos Final 27
3.1.9. Presupuesto Total Final 28
3.1.10. Control del Presupuesto 28
3.2. Sostenibilidad y Compromiso Social **29**
3.2.1. Encuesta de Sostenibilidad 29
3.2.2. Puntuación Sostenibilidad 30
3.2.3. Área Económica 32
3.2.4. Área Social 32
3.2.5. Área Ambiental 33
3.3. Leyes y regulaciones **33**
4. Aprendizaje por Refuerzo **34**
4.1. Redes Neuronales **37**
4.2. Algoritmos **40**
4.2.1. Q-Learning 40
4.2.2. SARSA 41

4.2.3. Actor-Critic (A2C) 42
4.3. Experience Replay 44
5. Escenarios de Entrenamiento 45
5.1. Harvest Wood 45
5.1.1. Descripción y objetivos 45
5.1.2. Elementos a tener en cuenta 46
5.1.3. Comportamiento del jugador 46
5.1.4. Recompensas 46
5.2. Chase the Player in the Labyrinth 48
5.2.1. Descripción y objetivos 48
5.2.2. Elementos a tener en cuenta 48
5.2.3. Comportamiento del jugador 49
5.2.4. Recompensas 49
5.3. Survive the Deadly Road 50
5.3.1. Descripción y objetivos 50
5.3.2. Elementos a tener en cuenta 51
5.3.3. Comportamiento del jugador 51
5.3.4. Recompensas 51
5.4. Fight Zombies 52
5.4.1. Descripción y objetivos 52
5.4.2. Elementos a tener en cuenta 52
5.4.3. Comportamiento del jugador 53
5.4.4. Comportamiento de los zombis 53
5.4.5. Recompensas 54
6. Definición del Estado y Acciones 55
6.1. Estado 55
6.2. Acciones 58
7. Implementación 61
7.1. Implementación dentro de Minetest 61
7.1.1. Conexión de Minetest 61

7.1.2. Personajes del jugador, agente aliado y zombis 61
7.1.3. Envío de mensajes al chat 63
7.1.4. Comandos y funciones 65
7.2. Implementación de la Simulación 67
7.2.1. Mundo de Minetest 67
7.2.2. Jugadores 70
7.2.3. Escenarios de entrenamiento 71
7.2.4. Agentes 74
7.2.5. Red neuronal 76
7.2.6. Algoritmos de aprendizaje por refuerzo 76
8. Agentes Q-Learning 77
8.1. Harvest in the Wood 78
8.1.1. Entrenamiento 78
8.1.2. Resultados 79
8.2. Chase the Player in the Labyrinth 80
8.2.1. Entrenamiento 80
8.2.2. Resultados 81
8.3. Survive the Deadly Road 82
8.3.1. Entrenamiento 82
8.3.2. Resultados 83
8.4. Fight Zombies 84
8.4.1. Entrenamiento 84
8.4.2. Resultados 85
9. Agentes SARSA 87
9.1. Harvest in the Wood 87
9.1.1. Resultados 87
9.2. Chase the Player in the Labyrinth 89
9.2.1. Resultados 89
9.3. Survive the Deadly Road 90
9.3.1. Resultados 90

9.4. Fight Zombies **91**
9.4.1. Resultados 91
10. Agentes A2C **93**
10.1. Harvest in the Wood **93**
10.1.1. Resultados 93
10.2. Chase the Player in the Labyrinth **94**
10.2.1. Resultados 94
10.3. Survive the Deadly Road **94**
10.3.1. Resultados 94
10.4. Fight Zombies **94**
10.4.1. Resultados 94
11. Conclusiones **95**
12. Opinión Personal **97**
13. Trabajo Futuro **99**
13.1. Probar mejoras de nuestros algoritmos **99**
13.2. Modificar nuestros escenarios/Añadir escenarios **99**
14. Bibliografía **100**

Índice de tablas y figuras

Tablas

1: Recursos materiales…….....17
2: Estimación de horas y dependencias inicial.…….....18
3: Estimación de horas y dependencias definitivo…...…….....19
4: Estimación de costes de recursos humanos…….24
5: Estimación de costes de hardware……...24
6: Estimación de costes de software…….....25
7: Estimación de costes indirectos……...25
8: Costes totales iniciales……..…26
9: Estimación de costes de recursos humanos final…….27
10: Costes totales finales……28
11: Matriz sostenibilidad……..……...30
12: Resultados de las pruebas de las configuraciones del estado……58
13: Resultados Q-Learning Harvest Wood lr=0.001……...79
14: Resultados Q-Learning Harvest Wood lr=0.0001……79
15: Resultados Q-Learning Chase the Player in the Labyrinth lr=0.001…….81
16: Resultados Q-Learning Chase the Player in the Labyrinth lr=0.0001……...81
17: Resultados Q-Learning Survive the Deadly Road lr=0.001……..83
18: Resultados Q-Learning Survive the Deadly Road lr=0.0001…….....83
19: Resultados Q-Learning Fight Zombies Road lr=0.001……85
20: Resultados Q-Learning Fight Zombies Road lr=0.0001……...85
21: Resultados SARSA Harvest Wood lr=0.001……87
22: Resultados SARSA Harvest Wood lr=0.0001……...88
23: Resultados SARSA Chase the Player in the Labyrinth lr=0.001……..89
24: Resultados SARSA Chase the Player in the Labyrinth lr=0.0001……89
25: Resultados SARSA Survive the Deadly Road lr=0.001……90
26: Resultados SARSA Survive the Deadly Road lr=0.0001……..91
27: Resultados SARSA Fight Zombies Road lr=0.001……...91
28: Resultados SARSA Fight Zombies Road lr=0.0001…….92
29: Resultados A2C Harvest Wood……93
30: Resultados A2C Chase the Player in the Labyrinth……..94
31: Resultados A2C Survive the Deadly Road……...94
32: Resultados A2C Fight Zombies…….94
33: Conclusiones Harvest Wood…….95
34: Conclusiones Chase the Player in the Labyrinth……...96
35: Conclusiones Fight Zombies……..…96

Figuras

1: Mundo de Minetest desde la vista del jugador……..…….13
2: Diagrama de Gantt inicial del proyecto……...21
3: Diagrama de Gantt definitivo del proyecto…….22
4: Esquema aprendizaje por refuerzo……..35
5: Red neuronal……....37
6: Funcionamiento de una neurona……..38
7: Función ReLU…….39
8: Función Softmax…….39
9: Pseudocodigo Q-Learning……...40
10: Pseudocódigo SARSA……...41
11: Como funciona Actor-Critic……..42
12: Proceso Actor-Critic 1……...43
13: Proceso Actor-Critic 2……...43
14: Proceso Acctor-Critic 3……44
15: Escenario de entrenamiento Harvest Wood en Minetest……...45
16: Escenario de entrenamiento Chase the Player in the Labyrinth en Minetest……48
17: Escenario de entrenamiento Survive the Deadly Road en Minetest…….50
18: Escenario entrenamiento Fight Zombies en Minetest……...52
19: Configuración 1 - posiciones que tiene en cuenta el agente……..57
20: Configuración 2 – posiciones que tiene en cuenta el agente…….57
21: Configuración 3 – posiciones que tiene en cuenta el agente…….57

1. Contexto y Alcance

1.1. Contexto

1.1.1. Introducción

En los últimos años, la Inteligencia Artificial (IA) ha crecido a una velocidad asombrosa y se considera que este mercado experimentará un crecimiento exponencial. Para comprobar este hecho con números, y no solo con palabras, el informe de la empresa de estudios de mercado Tractica [1] dice que es probable que crezca de 643,7 millones de dólares en 2016 hasta alcanzar los 36.000 millones de dólares en 2025. Lo que significa que en este periodo de tiempo el mercado de la IA se multiplicaría por 57.

En el pasado ver a un robot trabajar y desarrollar cierta inteligencia se había calificado de ciencia ficción. Incluso se han realizado cientos de películas con esa temática. Sin embargo, los robots, los coches sin conductor con un gran nivel de sofisticación y los últimos teléfonos móviles con estabilización de fotografías, entre muchos otros, son parte de nuestro día a día y nos hacen pensar lo prometedores que pueden llegar a ser en el futuro.

La inteligencia artificial y los juegos tienen una larga historia el uno junto al otro. Gran parte de la investigación de IA para juegos está basado en construir agentes para jugar a juegos, con o sin un componente de aprendizaje. Incluso antes de que la inteligencia artificial estuviera reconocida como un campo, los pioneros de ciencias de la computación escribían programas jugables como si fueran juegos porque querían testear si los ordenadores eran capaces de resolver tareas que parecían que necesitaban "inteligencia". Arthur Samuel fue el primero en inventar la forma de aprendizaje automático que hoy conocemos como aprendizaje por refuerzo utilizando un programa que aprendía a jugar al juego Checkers compitiendo contra si mismo.

¿Por qué los videojuegos son el dominio ideal para el estudio de la IA?

Los videojuegos ofrecen una de las formas más ricas de la interacción persona-computador. Esta riqueza está definida según las diversas opciones que tiene disponible un jugador en cualquier momento y de las diferentes formas o modalidades que el jugador puede interactuar con el medio. Las opciones disponibles para el jugador están asociadas al espacio de acción del juego y la complejidad asociada.

Los videojuegos ofrecen desafíos debido al esfuerzo y las habilidades que necesitan los jugadores para completarlos, o en el caso de los puzles, resolverlos. Eso hace que presenten problemas complejos debido a sus espacios de estados finitos, y las posibles estrategias de un agente, son normalmente extensos, a la vez que los espacios de soluciones pueden ser pequeños. Esto hace que sean realmente interesantes para el uso de IA en ellos.

Para darte a conocer al mundo necesitas realizar una aplicación sobre un tema popular en vez de uno que no se conoce mucho. Como todos sabemos los videojuegos tienen una gran popularidad en la actualidad, haciéndolos ideal como lugar donde poner en prueba a la IA y hacerla crecer en todo el mundo.

Además, la popularidad tiene otras ventajas aparte de poner a la IA en el escaparate de la actualidad ante todos. Primero, contra más personas juegan a videojuegos mayor contenido se necesita para crear un juego. Cuesta tiempo y esfuerzo crear contenido, pero a lo largo de los años se han desarrollado mecanismos que permite a jugadores y maquinas diseñar y crear varias formas de contenido en los juegos. Segundo, debido al gran número de personas que juegan se produce una generación masiva de datos sobre partidas y el comportamiento de los jugadores. Ante estas dos ventajas la inteligencia artificial puede ayudar a crear contenido y aparte obtiene una gran cantidad de datos que son beneficiosos para mejorar las IA en juegos específicos.

¿Por qué el uso de IA en videojuegos?

Aparte de motivos como la creación de contenido y la obtención de datos sobre el comportamiento de los jugadores, comentados en los apartados anteriores, hay otra razón muy importante que hay que tener en cuenta: la IA juega y mejora los juegos. [2]

Dentro de este contexto, de tanta importancia en la actualidad, es donde he decidido que realizaría mi trabajo de final de grado. En este proyecto utilizaré la rama de la inteligencia artificial llamada aprendizaje por refuerzo por tal de obtener una IA que ayude al jugador en el videojuego Minetest.

1.1.2. Formulación del problema

Normalmente la industria de los videojuegos es elogiada por la IA de sus juegos, es decir, por los personajes no controlables, también conocidos como NPC (*Non-Player Character),* o la IA del adversario que compite contra el jugador. Esto hace que la IA del juego añada valor al mismo contribuyendo a recibir mejores críticas y mejorando las experiencia de los jugadores. [2]

Sin embargo, programar una IA es complejo, y lo es más cuando tiene que abarcar todas las acciones o funciones necesarias para el videojuego e incluso actuar como lo haría un jugador. Por este motivo también puede quitarles valor a los juegos. Una IA demasiado fácil acabará aburriendo al jugador ya que no supondrá ningún reto pare él, un NPC aliado que no realice ninguna función útil, como ayudar al jugador a recoger materiales o a eliminar enemigos, hará que el jugador se sienta decepcionado y piense que colaborar con su aliado es inútil. Ambos casos crearían frustración en los jugadores llegando a tener consecuencias de que no vuelvan a jugar al juego e incluso no lleguen a comprar el próximo, produciendo perdidas evidentes en las compañías de videojuegos.

1.1.3 Actores implicados

En este aparatado se definirán los actores implicados en el proyecto, es decir, toda aquella persona que esté relacionada con el proyecto o esté interesada u obtenga algún beneficio del mismo.

Director

El director de este proyecto es Javier Bejar Alonso, profesor e investigador del departamento de Ciencias de la Computación en la Universitat Politècnica de Catalunya (UPC). Su papel es el de supervisar por tal de determinar posibles errores o dificultades en el transcurso del proyecto. Además, guía, aconseja y ayuda al desarrollador para que el proyecto alcance los objetivos establecidos y que se cumplan las fechas de entrega.

Desarrollador

Este proyecto es el trabajo de final de grado de un estudiante de la FIB, el cual también es el desarrollador, ese soy yo. Voy a trabajar en todo lo que requiera el proyecto, es decir, la documentación, la investigación e búsqueda de información, el código, los test y las conclusiones.

Beneficiarios

Al finalizar el proyecto no se habrá creado un producto que vaya a comprar un usuario o una organización, pero eso no significa que no tenga beneficiarios. En concreto habrá dos:

El primero, se trata de la industria del videojuego. Que observa que ya no solo grandes compañías en el campo de la inteligencia artificial se dedican a realizar proyectos o crear nuevo contenido, sino que alumnos que están a punto de terminar el grado de ingeniería informática, como yo, también estamos interesados. Además, los avances que se obtengan en cuanto al aprendizaje por refuerzo pueden mostrar los resultados que se han obtenido usando estas técnicas y hagan plantearse a las compañías desarrolladoras utilizarlas para sus juegos.

El segundo, y no menos importante, son los jugadores. Ha habido veces que en un buen juego los personajes no controlables o aliados no cumplían un nivel mínimo de comportamiento por lo que acaban arruinando la experiencia del jugador y provocándole frustración. Con mejores IAs los jugadores se sentirán más arropados por sus aliados pudiendo confiar en ellos y enfrentaran mayores retos cuando compitan contra sus enemigos haciendo que su experiencia sea divertida, gratificante y llena de desafíos esperando a ser superados.

1.2. Estado del Arte

Aprendizaje por refuerzo

El aprendizaje por refuerzo consiste en aprender que acción tomar en cada situación por tal de maximizar la recompensa, la cual se trata de un valor numérico. Al agente, el encargado de realizar el aprendizaje, no se le dice que acciones tiene que elegir, pero debe descubrir que acciones tienen mayor recompensa utilizándolas. En los casos más interesantes tomar una acción no solo afecta la recompensa inmediata, sino que también afecta a la siguiente situación y a todas las recompensas que obtendremos a continuación.

En resumen y simplificado, en un momento o *step* t, el agente está situado en un estado s y escoge realizar una acción a de entre todas las disponibles en s. Una vez tomada la acción, el entorno le envía una recompensa r. De esta forma el agente aprende que acciones maximizan la recompensa total, o suma de recompensas obtenidas de cada iteración, y descubrirá la política óptima. Con política nos referimos al comportamiento del agente frente a la decisión de elegir una acción.

Las dos características más importantes que distinguen al aprendizaje por refuerzo de otras ramas de la IA son la búsqueda por prueba y error y la recompensa retrasada. La primera, debido a que el agente prueba las diversas acciones a ver cuál le otorga una mayor recompensa; Y la segunda, dado que es más importante obtener al final una mayor recompensa incluso si tenemos que sacrificar parte de recompensa inmediata tomando otra acción en un estado. [3]

Minetest

Minetest se trata de un videojuego libre, de tipo *voxel* y mundo abierto para Windows, Linux, FreeBSD, Mac OS y Android.

La forma de jugar al juego es siempre la misma: El jugador aparece en un mundo hecho de cubos/bloques. Como el mapa está hecho con estos cubos, estos cubos pueden ser quitados y se pueden poner donde el jugador quiera con total libertad. Muchos de esos cubos representan materiales, como la madera o el carbón, que el jugador podrá recolectar por tal de combinarlos y crear herramientas, como picos para cavar más rápido, armas, como espadas, entre otros items. Sin embargo, las partidas pueden llegar a ser mucho más complicadas y se pueden jugar tanto en solitario como online con tus amigos u otros miembros de la comunidad.

El concepto principal de Minetest es sobre el juego base poder añadirle *mods* por tal de que el jugador pueda jugar una partida con todas las características o añadidos que deseé, basados en el lenguaje de programación Lua. [4]

Actualmente existen empresas muy importantes que están utilizando aprendizaje por refuerzo en videojuegos con el objetivo de mejorar las IA ya existentes y observar que resultados pueden obtener al competir contra otras IAs o expertos en el juego. A continuación se mostraran ejemplos actuales de proyectos:

Microsoft y Minecraft: Project Malmo

La plataforma Malmo creada por Microsoft es una sofisticada plataforma para la experimentación con IA construida sobre Minecraft, y diseñada por tal de ofrecer soporte a la investigación de la inteligencia artificial.

La plataforma de Project Malmo consiste en un mod, para la versión de java, y código que ayuda a los agentes de la inteligencia artificial a percibir y actuar en el entorno que ofrece el videojuego Minecraft.

Minecraft es ideal para la investigación de la inteligencia artificial debido a que ofrece a los usuarios infinitas posibilidades de como jugarlo. Pueden hacer cosas tan simples como ir de pesca o pasear por el bosque en busca de un árbol con manzanas, pero también permite realizar cosas más complejas como podría ser construir un parque de atracciones con un grupo de amigos o volar por los aires una cueva a base de poner TNT.

Project Malmo es libre, por lo que se puede encontrar en el github de Microsoft. Además, en 2017 Microsoft organizó la primera competición utilizando Malmo por tal de que los usuarios que quisieran participar hicieran equipos para que desarrollaran y entrenaran IAs que obtuvieran altas puntuaciones ante los retos propuestos en la competición. La segunda competición de Malmo se celebrará en este 2018 y sus inscripciones están abiertas. [5]

DeepMind (Google) y Starcraft 2

El objetivo principal de DeepMind es llevar a la inteligencia artificial más allá de sus límites. La forma de conseguirlo es desarrollando sistemas que puedan resolver problemas complejos. Anteriormente han diseñado agentes para comprobar su comportamiento en juegos como Atari y Go, pero finalmente dieron un gran paso al decidir desarrollar agentes para unos de los juegos más exitosos y complejos de los últimos tiempos, Starcraft II.

Aunque el objetivo de la serie Starcraft ha sido siempre derrotar a uno o varios oponentes, la verdadera dificultad siempre ha sido como balancear y alcanzar en el momento adecuado varios subobjetivos. Algunos ejemplos serían: recoger recursos, construir estructuras y realizar investigaciones, entre otros. Además, el mapa es solo parcialmente observable, ya que el jugador solo puede ver aquella zona sobre la que está situada la cámara, lo que hace que los agentes tienen que combinar memoria y planificación.

Starcraft es realmente interesante para la IA por dos motivos. El primero, hay una gran cantidad de jugadores online cada día, lo que permite obtener grandes cantidades de datos que se pueden utilizar para el aprendizaje. El segundo, el espacio de acción es realmente complejo ya que las acciones disponibles en cualquier momento son más de 300. [6]

Conclusiones

Como podemos observar, la IA y el aprendizaje por refuerzo se están utilizando en gran medida en la actualidad en el campo de los videojuegos. Ya no solo en proyectos de final de carrera, como el mío, sino que también grandes empresas como DeepMind y Microsoft utilizan el aprendizaje por refuerzo en videojuegos por tal de seguir investigando la inteligencia artificial y superar los límites actuales.

Otra conclusión que obtengo, y muy importante, es el haber elegido correctamente para este proyecto el juego de *voxel* Minetest. Esto es debido a que Microsoft está utilizando aprendizaje por refuerzo en Minecraft, un juego que es idéntico a Minetest, por lo que significa que se trata de un videojuego interesante para aplicar estos métodos, entrenar agentes y obtener datos y resultados.

1.3. Objetivos

El objetivo principal de este proyecto es obtener un aliado NPC *(Non-Player Character)*, en el juego Minetest, que alcance un buen comportamiento para llegar a ser útil al jugador. El aliado será una IA y se utilizará la rama de la inteligencia artificial llamada aprendizaje por refuerzo por tal de obtener el comportamiento mencionado anteriormente. El aprendizaje por refuerzo hará que el agente, el encargado de realizar el aprendizaje, aprenda cual es la mejor acción a realizar en cada situación.

Para poder alcanzar este objetivo hay subobjetivos en este proyecto que hay que cumplir satisfactoriamente:

Para empezar, hay que realizar una buena selección y diseño en los escenarios específicos donde se entrenaran los agentes. Se tienen que elegir escenarios en los que el agente pueda entrenar tareas comunes en el juego de Minetest y realizar un diseño adecuado para su aprendizaje.

Por otra parte, los agentes que utilizarán diferentes métodos de aprendizaje por refuerzo deberán alcanzar un nivel alto y óptimo en cada uno de los escenarios de aprendizaje que contendrán diferentes aspectos del juego. Los agentes aumentaran su nivel de comportamiento conforme realicen ejecuciones en cada uno de los escenarios de entrenamiento.

Además, otro subobjetivo muy importante sería que los algoritmos de aprendizaje por refuerzo (Q-Learning, Sarsa, A3C,…) y los scripts con toda la información necesaria del entorno y parámetros del agente estarán en el lenguaje de programación Python y serán externos a Minetest. Estos scripts deben de poder comunicarse entre ellos y con la API de Minetest por tal de poder enviar información al juego y poder realizar simulaciones.

1.4. Alcance

Para empezar, fue necesaria la instalación de JetBrains PyCharm Edu por tal de obtener un entorno de trabajo de Python por tal de utilizar aprendizaje por refuerzo. A continuación, hizo falta configurar este entorno por tal de establecer una conexión con la API de Minetest.

Tras el primer paso, se diseñaron diferentes escenarios en los cuales se entrenarían a los agentes. Estos escenarios son concretos y sus objetivos serán la de recrear varias situaciones que se encontrará un jugador de Minetest en una partida real y requerirán de distintas acciones y habilidades para superarlos. Un ejemplo sería un escenario donde la IA tiene que recolectar el mismo tipo material que el jugador por tal de ayudarle con la tarea de recolección.

En estos escenarios se entrenaran diferentes agentes, uno para cada método de aprendizaje por refuerzo que utilicemos, y se observarán y estudiaran los resultados que obtendrán las funciones de refuerzo por tal de determinar cual tiene un mejor comportamiento.

Finalmente, se escogerá la IA cuyo agente haya obtenido mejores resultados y la añadiremos a un personaje en un mundo abierto y recién creado de Minetest. De esta forma veremos su comportamiento ante un mundo real y podremos observar si los varios escenarios de entrenamiento han sido suficientes o si hubiera sido bueno añadir alguno más que no hubiéramos tenido en cuenta anteriormente en la parte de diseño de escenarios.

1.5. Posibles Obstáculos

Conexión con Minetest

Minetest nos permitirá visualizar el comportamiento del agente viendo a un personaje más jugando. En caso de que esta conexión falle y no nos deje pasar los elementos necesarios desde el Python, será imposible visualizar la simulación.

Tiempo y planificación

El tiempo que tenemos para este proyecto es limitado y específicamente se trata de 4 meses. Como todos sabemos, un obstáculo o problema en un proyecto software fácilmente puede llegar a consumir una gran cantidad de tiempo para solucionarlo y así desviarse de la planificación inicial.

Comportamiento final de la IA

Este proyecto tiene como finalidad obtener una IA que aplicada sobre un personaje ayude, o sea de utilidad, al jugador. Si después de todo el entrenamiento obtenemos como resultado un comportamiento muy lejos de lo óptimo, la prueba final de introducir al personaje en un mundo real del juego no será de gran interés.

Tiempo de entrenamiento

He considerado este obstáculo como distinto del de tiempo y planificación ya que este se refiere a una dificultad del aprendizaje por refuerzo que es el tema principal del proyecto y no a un problema cualquiera.

Los agentes requieren de un tiempo de entrenamiento muy elevado. Debido a ello en el caso de que haya que realizar ajustes para mejorar el comportamiento o volver a repetir algún escenario, puede que en el proyecto no se puedan usar tantos métodos de aprendizaje por refuerzo como eran esperados o que algún agente se quede incompleto al no haber podido completar todos los escenarios de entrenamiento.

1.6. Metodología y Rigor

Ciclos de desarrollo cortos

El metodología empleada para este proyecto es la *Agile* donde el trabajo ha sido repartido en ciclos cortos. Los objetivos se especificarán semanalmente en las reuniones con el director del proyecto y, a un nivel superior, se ha trabajado en ciclos cortos tratando de realizar todas las tareas del proyecto. Los ciclos se han distribuido de la siguiente forma:

1. Análisis del proyecto y tomar decisiones de realización (Decidir si usar un entorno de programación, que lenguaje de programación utilizar, como aplicaremos el aprendizaje por refuerzo,…)
2. Instalación y configuración de Minetest con PyCharm Edu
3. Selección, diseño e implementación de los escenarios de entrenamiento

4. Utilización de métodos de aprendizaje por refuerzo en agentes en los escenarios específicos y análisis de resultados

***Feedback* intensivo por parte del director**

En cuanto al seguimiento y la validación, se han concertado reuniones con el director semanalmente por tal de evaluar el estado del proyecto, ver si avanza adecuadamente y recibir consejos y *feedback* por tal de evitar obstáculos y posibles contratiempos que comprometan la planificación del proyecto.

1.7. Análisis de la Metodología y Rigor

Tanto la metodología, como el seguimiento y la validación explicados anteriormente no han variado en el transcurso del proyecto. Esta forma de trabajar ha resultado efectiva y ha permitido al proyecto avanzar adecuadamente en todos los aspectos.

1.8. Integración de Conocimientos

- **APA (Aprendizaje Automático):**

  La asignatura de APA me proporcionó conocimientos sobre *machine learning*. Sin embargo, pese a ser una rama diferente de la IA que el aprendizaje por refuerzo, me proporcionó el conocimiento de que es una red neuronal y realizamos laboratorios y ejercicios donde las utilizábamos. En mi proyecto también trabajaré con redes neuronales. Por ejemplo: en APA entrenábamos redes neuronales para predecir valores desconocidos a partir de datos que conocíamos, mientras que en mi proyecto utilizaré las redes neuronales para que el agente realice las decisiones de sus acciones.

- **A (Algoritmia):**

  La asignatura de A me ha proporcionado conceptos algorítmicos básicos y otros conceptos algorítmicos y de programación más avanzados como los algoritmos voraces. En mi proyecto trabajaré con algoritmos de aprendizaje por refuerzo por lo que estos conocimientos serán útiles. Por ejemplo: En A realizamos varios programas que resolvíamos vorazmente, como el problema de la mochila, mientras que el aprendizaje por refuerzo también dispone de métodos voraces ya que los agentes deben obtener los mejores resultados posibles.

- **LP (Lenguajes de programación):**

  La asignatura de LP me ha proporcionado conceptos básicos sobre los lenguajes de scripting. Realizamos una breve introducción a Python, mediante laboratorios y ejercicios de jutge, y asistí a una presentación donde se mostraban las características más importantes y básicas de Lua. Este proyecto se programará en Python y hará falta un conocimiento básico de Lua por tal de entender, realizar cambios e implementar el código de Minetest.

2. Planificación Temporal

2.1. Programa

El proyecto ha tenido una duración aproximada de 4 meses, teniendo inicio a mediados de septiembre y finalizando a finales de enero, aunque durante el mes de julio trabajé en la parte de investigación debido a su complejidad e importancia en el proyecto.

Respecto a la planificación, ha sido imposible que fuera fija. Este proyecto tiene diversas complejidades, que en el momento que algo no funcionara como era esperado habría que volver a investigar, y también se llevan a cabo experimentos, el entrenamiento de las IAs, que dependiendo del algoritmo y escenario el tiempo de entrenamiento puede variar en gran medida.

2.2. Descripción de las Tareas

2.2.1. Aprender sobre el aprendizaje por refuerzo

Aun habiendo superado la asignatura de IA, hace unos cuatrimestres, en el temario no se trata nada sobre el aprendizaje por refuerzo. Así que el primer objetivo de este proyecto ha sido obtener más conocimientos sobre el tema y familiarizarme con conceptos base.

Empecé viendo el curso de Google Deepmind [7] sobre aprendizaje por refuerzo, donde se trata desde conocimientos básicos a más avanzados, y utilicé como soporte a este curso el libro *Reinforcement Learning: An Introduction* [3], en el cual podía buscar y ampliar la información sobre cualquier tema, que no terminara de entender o que me surgiera un gran interés, del curso.

Además, leí un libro recomendado por mi director titulado *Artificial Intelligence and Games* [2]. De este libro no solo obtuve conceptos de aprendizaje por refuerzo, sino que aprendí sobre la historia de la IA y los juegos y sobre los métodos que utilizan las IAs para enfrentarse a diferentes tipos de juegos.

Esta tarea ha sido la primera y se ha realizado durante el mes de julio y las primeras semanas de septiembre.

2.2.2. Instalación y configuración del entorno

Una vez decidí con mi director de TFG de que haría el proyecto, lo primero que hice fue bajarme la versión de Minetest más actual (0.4.17.1) por tal de trabajar siempre con la misma versión y que los avances que realizáramos siempre fueran compatibles, en vez de estar actualizando cada mes. Más adelante, fue cuando pensé que sería mejor instalarse un entorno de desarrollo especializado en Python que utilizar un editor de texto. Decidí usar el IDE (Integrated Development Environment) de JetBrains llamado PyCharm Edu ya que aparte de ofrecerte muchas facilidades para programar en Python también tiene la opción de cargar cursos sobre el mismo lenguaje de programación por si surgiera cualquier duda. La versión de Python que se utilizará en este proyecto es la 3.6.

Esta tarea se ha realizado en paralelo a la tarea de **aprender sobre el aprendizaje por refuerzo.**

2.2.3. Familiarización con Minetest y su conexión con JetBrains PyCharm Edu

El mundo de Minetest está formado por cubos/bloques, pero dependiendo de lo que representen tendrán unas propiedades u otras, es decir, no todos los bloques son iguales ni tienen las mismas características. Por ejemplo el personaje puede recoger arena del suelo, pero para coger roca necesitará la herramienta pico para recolectarla. Como el mundo está formado por bloques, los materiales que obtiene el jugador se guardan en su inventario y se contabilizan en número de bloques. Pasemos a ver que vería un jugador en una partida:

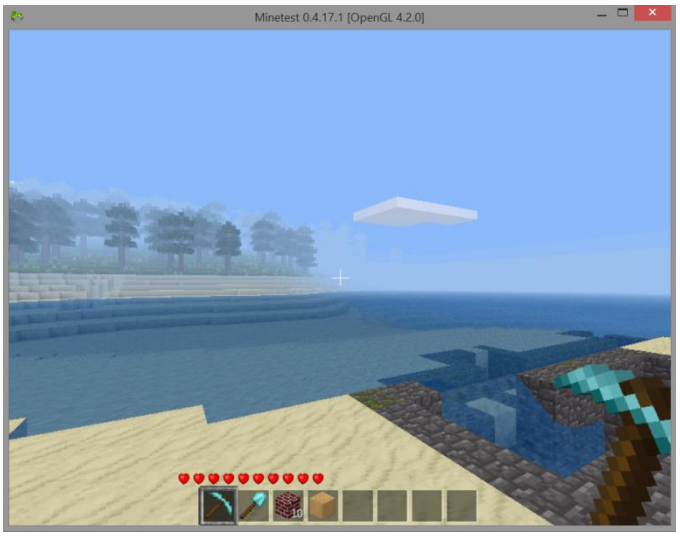

Figura 1: Mundo de Minetest desde la vista del jugador

En la figura 2.1 podemos ver perfectamente como el mundo está hecho de bloques, tal y como hemos explicado anteriormente, e incluso las nubes están formadas de esta forma. Además, podemos observar en la figura otros elementos importantes en la parte inferior de la imagen. Los corazones, representan la vida o puntos de salud del personaje y disminuirán cuando sea atacado por algún enemigo o caiga desde una gran altura. Los ocho cuadrados de una tonalidad de gris representa el inventario del personaje, en estos espacios el jugador podrá guardar aquellos materiales o herramientas que deseé. Como podemos ver en la figura, el jugador dispone de un pico, una pala, doce cubos de ladrillo y un cubo de arena. Es interesante ver como cubos del mismo tipo no van a otro espacio del inventario, sino que se añade un número, tal y como pasa con los ladrillos, y también cabe destacar como el elemento que tiene equipado o seleccionado el jugador aparecerá en la parte derecha. En caso de no tener nada equipado, se verá el brazo del personaje que está formado también por cubos, pero estos más pequeños.

El código de Minetest está realizado en el lenguaje de programación Lua. Sin embargo, en este proyecto trabajamos con Python de forma externa al código Lua. Se ha tenido que establecer una conexión con el IDE PyCharm Edu por tal de que cuando realicemos un programa este pueda comunicarse con Minetest y ejecute las funciones correspondientes. También forma parte de la familiarización con Minetest el mirar su código y ver como se definen los tipos de cubos y herramientas internamente por si necesitamos usarlos en el Python llamarlos de la misma forma y mantener la coherencia.

Esta tarea se ha realizado a continuación de las tareas de **aprender sobre el aprendizaje por refuerzo** e **instalación y configuración del entorno**.

2.2.4. Diseño y selección de los escenarios de entrenamiento

Como se ha mencionado anteriormente, los agentes de este proyecto realizaran su aprendizaje en escenarios específicos que podrían darse en una partida real de Minetest. Por lo tanto, era muy importante definir cuáles serían estos escenarios. Los agentes serán entrenados en los mismos escenarios, pero estos dispondrán de un componente aleatorio o *random* para su creación por tal de que no aprendan mediante repetición, sino que realmente tengan que explorar el escenario por tal obtener el mejor resultado.

Los escenarios se han planteado de forma que el agente deba realizar tareas distintas en cada uno de ellos. De esta forma en cada escenario entrenará una parte del comportamiento que necesitaría para jugar en un mundo de Minetest real. Esto nos permite observar cómo se comporta la IA frente a los diferentes tipos de problemas. Por ejemplo una de las IAs puede ser mejor en el escenario de combate enfrentándose a enemigos, pero no llega a ser tan buena en el escenario de recolección de materiales.

Esta tarea se ha realizado después de la tarea de **familiarización con Minetest y su conexión con JetBrains PyCharm Edu**.

2.2.5. Implementación, entrenamiento, evaluación del rendimiento de los agentes y prueba final

Puede que esta sea una de las partes más importantes del proyecto, la tarea donde se implementa y se entrena la IA. Se han implementado agentes seleccionando diversos algoritmos de aprendizaje por refuerzo. Estos agentes entrenaron realizando ejecuciones en cada uno de los escenarios específicos diseñados con anterioridad. Tras diversas ejecuciones, se realizaron pruebas por tal de ver como avanzaban sus comportamientos y, en caso de no ser muy satisfactorio el avance, se realizaron los ajustes necesarios.

Dado que el objetivo de este proyecto es la creación de una IA que sea un buen aliado para un jugador de Minetest, una vez finalizado el entrenamiento se evaluaron los diferentes agentes y se efectuó la prueba final introduciendo el agente en un mundo de Minetest. Como se ha comentado anteriormente, esta prueba consiste en introducir al agente en un mundo real de Minetest y comprobar si todo su entrenamiento ha servido para desarrollar un buen comportamiento y ser útil al jugador.

Esta tarea se ha realizado en paralelo de la tarea **diseño y selección de los escenarios de entrenamiento.**

2.2.6. Análisis y conclusiones

Se analizaron los diferentes resultados de los algoritmos y se sacaron conclusiones sobre ellos comparando unos con otros.

Esta tarea se ha en paralelo de la tarea **implementación, entrenamiento, evaluación del rendimiento de los agentes y prueba final.**

2.2.7. Memoria del proyecto

La memoria del proyecto es la documentación realizada tanto en GEP como a continuación hasta que finalizara el proyecto.

Esta tarea se ha realizado en paralelo a todas las tareas del proyecto.

2.2.8. Preparación de la presentación final del proyecto

Antes de realizar la defensa del proyecto, se tuvo que realizar las diapositivas de la presentación y prepararlas. Esta tarea se ha realizado una vez finalizada la tarea de **memoria del proyecto**.

2.3. Plan de acción y valoración de alternativas

No suficiente potencia de cómputo: tendría que hablar con el director y la FIB para ver si me pudiesen proporcionar un ordenador con mayor potencia.

La conexión con Minetest falla: si llegará la situación de no poderse arreglar, podríamos utilizar el código libre de Project Malmo y realizar los experimentos que nos diera tiempo antes de la entrega final.

El comportamiento final de la IA no es satisfactorio: si fallaran todos los algoritmos elegidos habría que buscar uno nuevo y realizar los entrenamientos que fueran posibles con el tiempo que tuviéramos.

Utilización de sistemas de computación en la nube: Tanto Amazon con AWS (Amazon Web Services) como Google, disponen de servicios de computación en la nube con mayor potencia que mis recursos o los que me pudiera ofrecer la FIB.

2.4. Análisis actual de las alternativas

Potencia de cómputo: Después de realizar varias iteraciones de entrenamientos de los agentes, podemos decir que mi portátil nos permite realizar dichos entrenamientos con buenos tiempos, sin llegar a ser excesivo.

Conexión con Minetest: A este aspecto se le ha tenido que dedicar más tiempo de lo planificado, pero hemos conseguido establecer la conexión tanto para poder crear nuestros escenarios en el juego como para poder observar a los agentes en simulaciones.

Comportamiento de la IA: Estamos en una fase temprana de los entrenamientos y aún se pueden realizar bastantes ajustes en los parámetros de entrenamiento o los escenarios. Por lo que comentar si tiene un buen comportamiento o no ahora mismo sería precipitarse.

Sistemas de computación en la nube: Estamos barajando de utilizar esta opción, pero habrá que tener en cuenta los requisitos necesarios y si habría que realizar alguna instalación de software en mi portátil personal.

2.5. Recursos

Portátil personal	Para trabajar en casa, dispongo de un ordenador portátil.
JetBrains PyCharm Edu	Entorno de desarrollo de Python.
Librerías python aprendizaje por refuerzo	Librerías con los algoritmos necesarios para realizar el entrenamiento de agentes.
Minetest	Videojuego libre en el que se basa este proyecto.
Modds Minetest	Minetest es bastante básico por lo que necesita algunos añadidos por tal de realizar escenarios más reales.
Microsoft Word	Para realizar la memoria del proyecto.
LaTeX	Para realizar las diapositivas del proyecto.
Github	Para tener un repositorio del proyecto que permite tanto tener una copia de seguridad como poder ver los avances del mismo a lo largo del tiempo.

Tabla 1: Recursos materiales

2.6. Estimación de Horas y Dependencias Inicial

Tarea	Horas	Dependencias
1)Aprender sobre aprendizaje por refuerzo	80	-
Lecturas	50	
Curso Google Deepmind	30	
2)Instalación y configuración del entorno	1	Paralelo 1)
Instalar Minetest y PyCharm Edu	1	
3)Familiarización con Minetest y su conexión con PyCharm Edu	69	1) 2)
Minetest	10	
Conexión Pycharm Edu	59	
4)Diseño y selección de los escenarios de entrenamiento	35	3)
5)Implementación, entrenamiento, evaluación del rendimiento de los agentes y prueba final	160	4)
Implementación	90	
Evaluación del rendimiento	50	
Prueba final	20	
6)Análisis y conclusiones	30	Paralelo 5)
7)Memoria del proyecto	135	Paralelo todas tareas
GEP	90	
Resto de memoria	45	
8)Preparación de la presentación final del proyecto	30	7)
Director de proyecto	80	-
TOTAL	**620**	

Tabla 2: Estimación de horas y dependencias inicial

2.7. Estimación de Horas y Dependencias Final

Tarea	Horas	Dependencias
1)Aprender sobre aprendizaje por refuerzo	80	-
Lecturas	50	
Curso Google Deepmind	30	
2)Instalación y configuración del entorno	1	Paralelo 1)
Instalar Minetest y PyCharm Edu	1	
3)Familiarización con Minetest y su conexión con PyCharm Edu	69	1) 2)
Minetest	10	
Conexión Pycharm Edu	59	
4)Diseño y selección de los escenarios de entrenamiento	35	3)
5)Implementación, entrenamiento, evaluación del rendimiento de los agentes y prueba final	220	Paralelo 4)
Implementación	100	
Evaluación del rendimiento	100	
Prueba final	20	
6)Análisis y conclusiones	60	Paralelo 5)
7)Memoria del proyecto	135	Paralelo todas tareas
GEP	90	
Resto de memoria	45	
8)Preparación de la presentación final del proyecto	30	7)
Director de proyecto	80	-
TOTAL	**710**	

Tabla 3: Estimación de horas y dependencias definitivo

2.7.1. Cambios respecto la planificación inicial

Los cambios que ha habido respecto a la planificación inicial son los siguientes:

Dependencias de tareas

En un principio se planteaba el diseño y selección de los escenarios de entrenamiento antes de la tarea correspondiente a la implementación. Sin embargo, estas dos tareas han acabado haciéndose en paralelo. Es cierto que los escenarios han sido seleccionados con anterioridad, pero los escenarios se han diseñado y modificado conforme se han realizado sus implementaciones para ajustarlos a un nivel en que la IA pueda realizar su entrenamiento de forma óptima.

Esta variación no ha producido cambios ni en el número de horas de otras tareas ni en el coste del proyecto. Lo que sí que se nota significativamente en el diagrama de Gantt definitivo del proyecto ya que al realizar esta tarea en paralelo ha provocado que el resto de tareas se alargue en el tiempo como es el caso de la implementación o que se tenga que aplazar su comienzo a semanas posteriores como ha sucedido con la parte del resto de la memoria.

Evaluación del rendimiento y análisis del entrenamiento de los agentes

La planificación inicial suele ser la ideal, pero siempre surgen contratiempos en este tipo de proyecto. Durante la parte final del entrenamiento de los agentes, una vez se probaron dentro del juego de Minetest, se encontraron *bugs* que había que resolver.

Este contratiempo ha provocado que haya aumentado el tiempo tanto de evaluación de rendimiento como de análisis de los agentes en la planificación.

2.8. Diagrama de Gantt Inicial del Proyecto

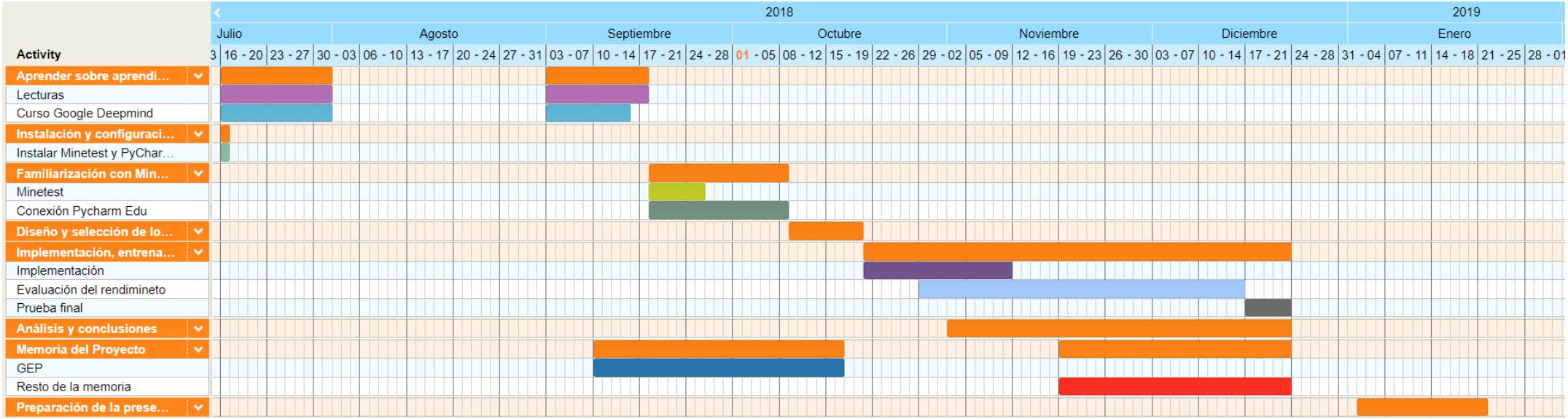

Figura 2: Diagrama de Gantt inicial del proyecto

2.9. Diagrama de Gantt Definitivo del Proyecto

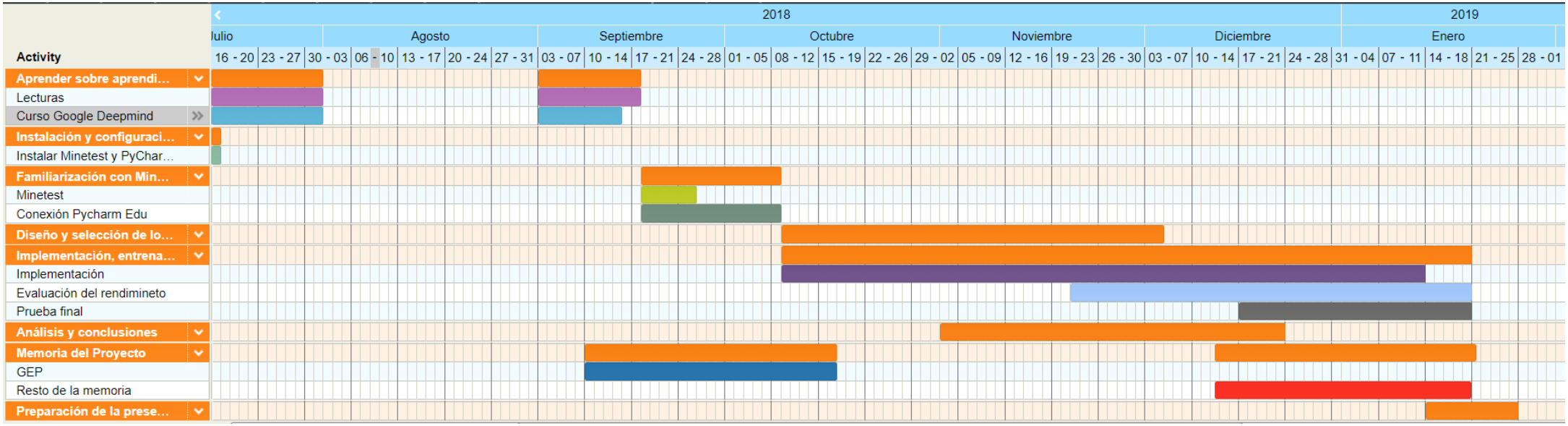

Figura 3: Diagrama de Gantt definitivo del proyecto

3. Coste y sostenibilidad

3.1. Gestión Económica

3.1.1. Consideraciones Iniciales

Para la realización de este proyecto se utilizarán todos los elementos detallados en el apartado de recursos comentados en el apartado anterior sobre **Planificación**. Estimaremos el coste del proyecto de los recursos humanos, hardware, software y costes indirectos.

3.1.2. Presupuesto de Recursos Humanos Inicial

Debido a que este proyecto es un TFG, los recursos humanos están compuestos por un estudiante, yo, y el director del TFG que pertenecemos a la universidad. A continuación analizaremos el coste de cada una de las partes del diagrama de Gantt y añadiendo el trabajo realizado por el director del proyecto. [8] [9]

Tareas	Horas	Precio	Precio Total
Director de proyecto	80	35€/h	2800€
Aprender sobre aprendizaje por refuerzo	80	10€/h	800€
Instalación y configuración del entorno	1	10€/h	10€
Familiarización con Minetest y su conexión con PyCharm Edu	69	10€/h	690€
Diseño y selección de los escenarios de entrenamiento	35	10€/h	350€

Tareas	Horas	Precio	Precio Total
Implementación, entrenamiento, evaluación del rendimiento de los agentes y prueba final	160	10€/h	1600€
Análisis y conclusiones	30	10€/h	300€
Memoria del proyecto	135	10€/h	1350€
Preparación de la presentación final del proyecto	30	10€/h	300€
TOTAL	**620**		**8200€**

Tabla 4: Estimación de costes de recursos humanos inicial

3.1.3. Presupuesto de Hardware

Para realizar la implementación del proyecto y su documentación son necesarios los elementos hardware. En este proyecto solo habrá un elemento hardware que será mi ordenador portátil personal.

Hay que tener en cuenta que el precio del hardware no se puede aplicar entero al proyecto ya que depende de la durada del mismo. Para calcular su precio verdadero relacionado con el proyecto o amortización se utilizará la siguiente formula:

Amortización = (Precio producto / Vida útil * Días laborables de un año (220) * Horas laborables en un día (8)) x horas del proyecto

Producto	Precio	Vida útil	Amortización
Portátil personal	700€	5 años	49,31€
TOTAL			**49,31€**

Tabla 5: Estimación de costes de hardware

3.1.4. Presupuesto Software

Respecto al software, se necesitarán las siguientes herramientas software para realizar este proyecto.

Producto	Precio	Vida útil
JetBrains Pycharm Edu	gratuito	-
Librerías Python aprendizaje por refuerzo	gratuito	-
Minetest	gratuito	-
Modds minetest	gratuito	-
Microsoft Word	69€	1 año
LaTex	gratuito	-
TOTAL	**69€**	

Tabla 6: Estimación de costes de software

3.1.5. Costes Indirectos

Los costes indirectos son aquellos fuera de los recursos humanos, software y hardware que se necesitan para realizar el proyecto.

Producto	Precio	Unidades	Precio Total
Electricidad	0,15€kW/h	1500 kWh	225€
Internet	50€/mes	4 meses	200€
Tarjeta Transporte	10,25€/mes	4 meses	41€
Material de oficina	-	-	15€
TOTAL			**481€**

Tabla 7: Estimación de costes indirectos

3.1.6. Imprevistos

En un proyecto de este tipo, es difícil realizar una estimación de los costes exacta. Existen muchos factores que en caso de suceder incrementarían los costes. Explicaré los principales a continuación:

- **Avería en el ordenador personal:** Solo dispongo de mi portátil para realizar el proyecto, así que en caso de que fallara habría que repararlo o substituirlo aumentando el coste del proyecto. Durante este tiempo podría pedir un ordenador a la universidad para no perder tiempo de proyecto.

Estimación en caso de avería:

- Reparación ordenador: 50-100€
- Asignación ordenador universidad: 0€ (gratuito para estudiantes)

- **Retraso en el desarrollo:** Podría retrasarse el proyecto debido a su complejidad y a que no disponemos de hardware especializado para realizar aprendizaje por refuerzo, esto haría que se incrementaran los gastos de recursos humanos y costes indirectos.

  Estimación de 1 mes de proyecto más:

  - R.R.H.H: 2050 €
  - Costes Indirectos: 120,25€

- **Uso de más hardware:** Aunque en principio no parece necesario, si el hardware actual asignado al proyecto no fuera suficiente para desarrollarlo, se tendría que contratar sistemas de computación en la nube lo que incrementaría los costes de hardware. Al ser el imprevisto menos probable por el momento no se ha realizado una estimación.

3.1.7. Presupuesto Total Inicial

Finalmente, observamos los costes totales del proyecto en una misma tabla. Aparte al coste total del proyecto se realizara la aplicación de una contingencia del 10 por ciento.

Concepto	Precio Estimado
RR. HH.	8200€
Hardware	49,31€
Software	69€
Indirectos	481€
Total	**8799€**
Total + 10% Contingencia	**9679€**

Tabla 8: Costes totales iniciales

3.1.8. Presupuesto de Recursos Humanos Final

Teniendo en cuenta los cambios en la planificación en presupuesto de recursos humanos sería el siguiente.

Tareas	Horas	Precio	Precio Total
Director de proyecto	80	35€/h	2800€
Aprender sobre aprendizaje por refuerzo	80	10€/h	800€
Instalación y configuración del entorno	1	10€/h	10€
Familiarización con Minetest y su conexión con PyCharm Edu	69	10€/h	690€
Diseño y selección de los escenarios de entrenamiento	35	10€/h	350€
Implementación, entrenamiento, evaluación del rendimiento de los agentes y prueba final	220	10€/h	2200€
Análisis y conclusiones	60	10€/h	600€
Memoria del proyecto	135	10€/h	1350€
Preparación de la presentación final del proyecto	30	10€/h	300€
TOTAL	**710**		**9100€**

Tabla 9: Estimación de costes de recursos humanos final

3.1.9. Presupuesto Total Final

Los costes totales solo han variado en el apartado de recursos humanos.

Concepto	Precio Estimado
RR. HH.	9100€
Hardware	49,31€
Software	69€
Indirectos	481€
Total	**9699€**
Total + 10% Contingencia	**10669€**

Tabla 10: Costes totales finales

3.1.10. Control del Presupuesto

En el control de presupuesto se utilizarán dos herramientas: la desviación en la realización de tareas en coste y en horas. Si fuese necesario debido a imprevistos, podríamos aplicar la desviación de coste de recursos hardware.

Desviación de realización de tareas (coste): Se calculará mediante la diferencia del coste estimado y el coste real y se multiplicará por el consumo de horas real.

Implementación, entrenamiento, evaluación del entrenamiento de los agentes y prueba final

(1600 – 2200)*220 = -132000

Análisis y conclusiones

(300 – 600)*60 = -18000

Desviación de realización de tareas (horas): Se calculará mediante la diferencia del consumo estimado y el consumo real y se multiplicará por el coste real.

Implementación, entrenamiento, evaluación del entrenamiento de los agentes y prueba final

(160 – 220)*2200 = -132000

Análisis y conclusiones

(30 – 60)*600 = -18000

Desviación de coste de recursos hardware: Se calculará mediante la diferencia del coste total estimado y el coste real total.

No ha hecho falta utilizar este estimador.

3.2. Sostenibilidad y Compromiso Social

3.2.1. Encuesta de Sostenibilidad

Una vez realizada la encuesta, veo que la sostenibilidad no se tiene uy en cuenta al hacer proyectos. Es cierto que no he trabajado ni he realizado ningún proyecto fuera de la universidad, sin embargo, en el momento de elegir el trabajo de final de grado o al realizar proyectos en las asignaturas no he tenido en cuenta si la sostenibilidad era buena o no. Es decir, ni ha estado presente a lo largo de la carrera al realizar proyectos. Como mucho podríamos encontrar algún tema sobre sostenibilidad en alguna competencia transversal donde tuvimos que asistir a una conferencia que nos hablaba sobre la sostenibilidad. Siempre el objetivo ha ido maximizar los beneficios sin tener en cuenta los impactos sociales o medioambientales.

Respecto a mis conocimientos, ha habido conceptos que sí que entendía, pero nunca había utilizado y otro que nunca había escuchado de ellos. Es cierto que podría identificar las causas de un problema relacionado con las TIC, pero realmente no sabría qué soluciones se han aplicado en otros casos similares a no ser que lo buscara por internet. Entiendo perfectamente que son las dimensiones medioambiental, social y económica, pero por otra parte no sé qué es tener en cuenta de forma holística, justicia social, equidad y diversidad. Por lo que muchas preguntas no he podido responde más favorablemente debido a que no sabía que efecto podría obtenerse de introducir uno de estos conceptos y entre otros. Otra pregunta que me ha parecido interesante es la de si soy capaz de introducir indicadores para estimas/medir. Es cierto, que hay conceptos que entiendo, pero no sé muy bien que indicadores se utilizan para realizar la estimación sobre la sostenibilidad.

Sobre la encuesta, he echado de menos un punto intermedio o neutro, ya que o estas bastante desacuerdo o bastante de acuerdo. Ya que evalúan nuestro conocimiento, podría haber alguna de las preguntas algún texto o información real por tal de que decidiéramos alguna de las respuestas, en vez de que fueran preguntas tan parecidas cambiando simples palabras. Quizás con contextos habría conceptos que se entendieran mejor.

En conclusión, la sostenibilidad en los proyectos es algo que no aparece en el Grado, excepto en alguna conferencia, hasta que he llegado a GEP. Si realmente es tan importante como se nos hace ver con la encuesta y tratando de ver la de nuestro proyecto de final de carrera creo que aparecería en más ámbitos de la carrera.

3.2.2. Puntuación Sostenibilidad

Dado que es un proyecto que se basa en la investigación, es difícil realizar un análisis de sostenibilidad al detalle con las preguntas de la tabla del documento de informe de sostenibilidad proporcionado para GEP. Se trata de una investigación sobre IA y aprendizaje por refuerzo que no se va a vender a la sociedad como un producto o servicio. Teniendo esto en cuenta, he intentado responder a las preguntas que he creído más relacionadas con este proyecto.

	PPP	**Vida Útil**	**Riesgos**
Ambiental	Consumo Diseño	Huella Ecológica	Riesgos Ambientales
	7/10	19/20	0/-20
Económico	Factura	Plan Viabilidad	Riesgos Económicos
	9/10	18/20	-2/-20
Social	Impacto Personal	Impacto Social	Riesgos Sociales
	9/10	19/20	0/-20
Rango Sostenibilidad	25/30	56/60	-2/-60
	79/90		

Tabla 11: Matriz de sostenibilidad

Consumo Diseño: El impacto será mínimo como hemos comentado anteriormente en el apartado de área ambiental y dado que solo se utiliza electricidad sería imposible reutilizarla. Por ello lo valoro con una nota de 7, ya que el impacto es mínimo, pero eso no significa que no haya consumo de electricidad o emisiones de CO_2 cada hora del proyecto.

Huella Ecológica: Mi proyecto solo utilizará un ordenador portátil para realizar todas las tareas. Por otro lado, Microsoft y DeepMind utilizarán una cantidad de electricidad superiormente exagerada debido a todos los equipos del personal y a los centros de súper computación que usan para realizar aprendizaje por refuerzo.

Riesgos Ambientales: En este proyecto no se puede dar ninguna situación que pudiera hacer que fuera más perjudicial al medioambiente ya que el impacto como hemos comentado anteriormente es mínimo.

Factura: Sí que se ha realizado una estimación de la realización del proyecto tanto de recursos humanos y materiales y se puede encontrar en el apartado anterior de **gestión económica.** Este apartado tiene una nota de 9 en la matriz de sostenibilidad ya que la factura de este proyecto sería bastante inferior a la que se obtendría si una empresa como Microsft lo realizara.

Plan Viabilidad: Respecto a soluciones existentes, mi solución con este proyecto es mucho más barata. En este caso, hay un desarrollador y director de proyectos, mientras que en Microsoft o DeppMind dispondrán de equipos con una cantidad más grande de personal y cada uno de ellos con su propio hardware y software necesario.

Riesgos Económicos: Los riesgos económicos son mínimos ya que si hubieran imprevistos estos no provocarían que el proyecto tardara mucho más en alcanzar la viabilidad o que dejase de ser rentable.

Impacto Personal: Aparte del conocimiento después de estudiar y trabajar con aprendizaje por refuerzo, este proyecto me aportará satisfacción y proyección laboral. Debido a que como he comentado anteriormente desde que hice la asignatura de IA me ha interesado mucho la inteligencia artificial y me gustaría trabajar en una empresa del sector. Es por estas razones que en la matriz de sostenibilidad le he dado una nota de 9.

Impacto Social: Generalmente se han programado IAs para controlar aliados en los juegos. Cuando un aliado no es de utilidad muchas veces perjudica al juego quitándole las ganas al jugador de continuar jugándolo, provocando pérdidas en ventas de ese juego o si hacen una continuación. Utilizar aprendizaje por refuerzo hará que los aliados aprendan del jugador y se adapten a su estilo de juego, provocando una sensación de utilidad y mejorando la experiencia de juego del jugador.

Riesgos Sociales: Este proyecto no podría presentar un impacto negativo en el mundo de los videojuegos y de la IA.

3.2.3. Área Económica

Para el área económica, se ha realizado un estudio detallado de los costes implicados en este proyecto como R.R.HH., hardware, software y costes indirectos, que se puede observar en el apartado anterior sobre **gestión económica**.

El proyecto en cuanto presupuesto es viable ya que será mucho más económico que otros proyectos sobre aprendizaje por refuerzo como pueden ser los de DeepMind con Starcraft II o Microsoft con Minecraft, ya que ellos al ser empresas tan potentes tienen un gasto mucho mayor en recursos humanos, hardware y software. Aparte de este gasto también tendrán un mayor impacto medioambiental debido al uso de estos recursos.

3.2.4. Área Social

Como se comentó al principio de la memoria, los beneficiarios de este proyecto serán los investigadores y empresas de inteligencia artificial y sobre todo aquellos que se enfocan en el aprendizaje por refuerzo. Proyectos de este tipo les proporcionaran más datos sobre los algoritmos que se utilizan y podrían observar cómo han sido diseñados los escenarios que han utilizado los agentes para realizar el entrenamiento y alcanzar su comportamiento final.

A nivel personal, soy un fan de los videojuegos y desde que cursé la asignatura de IA he estado muy interesado en ampliar mis conocimientos sobre la inteligencia artificial. Por lo que realizar este proyecto y estudiar sobre el aprendizaje por refuerzo para acabar aplicándolo, es una forma de cumplir ese objetivo y una ayuda para el momento en que tenga que buscar trabajo en alguna empresa del campo de la IA.

3.2.5. Área Ambiental

Por lo que respecta al área ambiental, este proyecto tiene un impacto mínimo, ya que el único recurso que utiliza que puede provocar un impacto ambiental es la electricidad necesaria para hacer funcionar el equipo. Es decir, realizar un estudio del impacto medioambiental de este proyecto no tendría mucho sentido.

También hay que tener en cuenta que los ordenadores emiten entre 52 y 234 gramos de CO_2 en una hora. Sin embargo, en este proyecto, solo se utilizará mi ordenador portátil y como máximo también el ordenador del director de proyecto en una reunión. Por lo que la emisión de CO_2 de 2 ordenadores será muy inferior a las emisiones que producirían grandes compañías como Microsoft teniendo en cuenta la cantidad de ordenadores y servidores que utilizarían. [10]

3.3. Leyes y regulaciones

La ley o regulación que se aplica a los videojuegos es el sistema PEGI. PEGI proporciona clasificaciones por edad para videojuegos en 38 países europeos. La clasificación por edad confirma que el juego es apropiado para jugadores de cierta edad, pero no tiene en cuenta el nivel de dificultad.

Minetest está clasificado como PEGI 7 debido a que su violencia no muestra daño o heridas a criaturas fantásticas. Además alguna de estas criaturas puede provocar miedo a niños por debajo de esa edad.

4. Aprendizaje por Refuerzo

Anteriormente, en el contexto del proyecto, se ha introducido el aprendizaje por refuerzo, pero en este apartado lo veremos en más detalle junto a otros elementos que se han utilizado en el proyecto.

Un ejercicio de aprendizaje por refuerzo está compuesto por dos componentes, un **agente (*agent*)** y un **entorno (*environment*)**. El entorno es aquel objeto con el que interactúa el agente, en este proyecto sería el videojuego Minetest, y el agente representa el algoritmo de aprendizaje por refuerzo.

Los conceptos básicos para entender el aprendizaje por refuerzo son:

- **Estado/*State* (s):** El estado almacena las características del entorno en un instante de tiempo y permite al agente saber en qué situación se encuentra dentro del mismo entorno.
- **Acción/*Action* (a):** Se trata del movimiento o actividad que realiza el agente en un instante de tiempo. La realización de una acción provoca que el estado cambie y pase del estado actual **s** al estado correspondiente **s'** tras haber realizado la acción **a**.
- **Recompensa/*Reward* (r):** Es la puntuación o bonificación que obtiene el agente tras realizar una acción en un estado. También pueden dar recompensas negativas por tal de penalizar ciertas acciones.
- **Política/*Policy* (π):** Se trata de la estrategia que sigue el agente para determinar la siguiente acción basada en el estado actual.
- ***Value* (V):** Se trata de la recompensa esperada a largo plazo con descuento, es el contrario a la recompensa r que es a corto plazo.
- **Q-Value o Action-Value (Q):** Q-value es similar a value, pero este recibe un parámetro extra, la acción actual a.

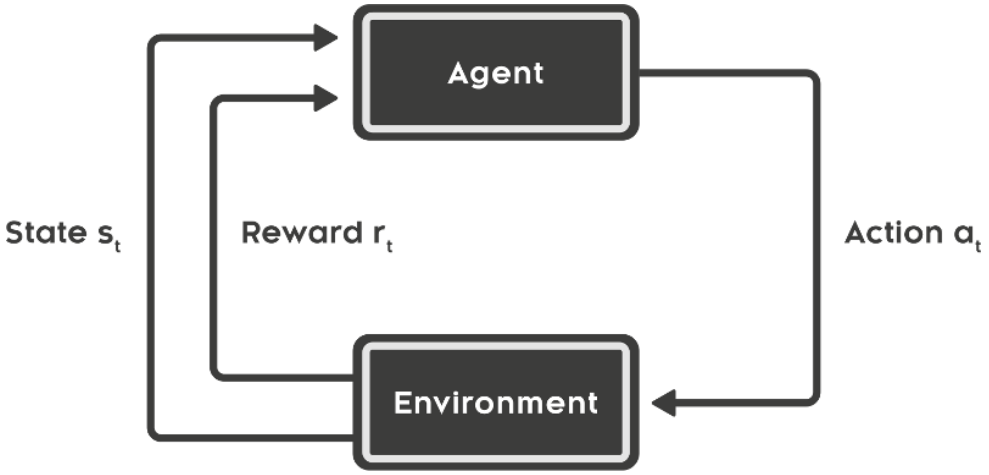

Figura 4: Esquema aprendizaje por refuerzo

En la figura 4 podemos ver con más claridad cómo funciona el aprendizaje por refuerzo y sus conceptos. Los pasos que se dan serían:

1. El agente observa el entorno obteniendo las características más importantes del mismo y se genera el estado, lo llamaremos **s1**.
2. El agente que se encuentra en el estado **s1** realiza la acción **a**.
3. Al realizar **a**, el agente modifica el entorno o su posición en él, por lo que recibirá la recompensa **r** que le indicará lo buena que ha sido la acción que ha realizado teniendo en cuenta que se encontraba en el estado **s1.** Al volver al paso 1, el agente obtendrá un nuevo estado fruto de la interacción del agente con el entorno.

Realizando estos pasos el agente tiene como objetivo obtener el mejor comportamiento, o también llamado como **política** en aprendizaje por refuerzo, por tal de saber en cada estado que acción tiene que realizar para maximizar la recompensa final. Este bucle continua indefinidamente hasta que el entorno le indica al agente que ha llegado a un estado terminal, el cual acaba el episodio.

Explotación vs Exploración

El balance entre la explotación y la exploración es uno de los desafíos que presenta el aprendizaje por refuerzo y que no se encuentra en otros tipos de aprendizaje.

Para obtener la recompensa máxima, el agente debe de realizar acciones que ya ha utilizado en el pasado y sabe que es efectiva en producir recompensa. Sin embargo, para llegar a descubrir estas acciones, ha tenido que probar acciones que no han sido seleccionadas anteriormente. El agente tiene que **explotar** lo que ha experimentado previamente para obtener recompensa, pero también tiene que **explorar** por tal de seleccionar mejores acciones en el futuro.

Si solo nos decidiéramos por utilizar una de las dos opciones, creyendo que es lo mejor, fallaríamos en la tarea de querer encontrar el comportamiento óptimo o de los mejores que el agente podría llegar a obtener. Lo ideal es balancearlas, que el agente pruebe una gran variedad de acciones y que progresivamente se favorezcan aquellas que parecen mejores.

On-policy vs Off-policy

Un algoritmo *on-policy* aproxima la política teniendo en cuenta las acciones del agente. Por otro lado, un algoritmo, *off-policy* utiliza la estrategia óptima a partir de una política definida que es independiente de las acciones del agente.

4.1. Redes Neuronales

Las redes neuronales [11] son dispositivos de cómputo modeladas a partir del funcionamiento del cerebro humano. Las redes neuronales han obtenido una gran importancia en las investigaciones de *Machine Learning* y en la industria debido a los avances que se han obtenido en el reconocimiento de voz, el procesamiento de texto y en la visión por computación.

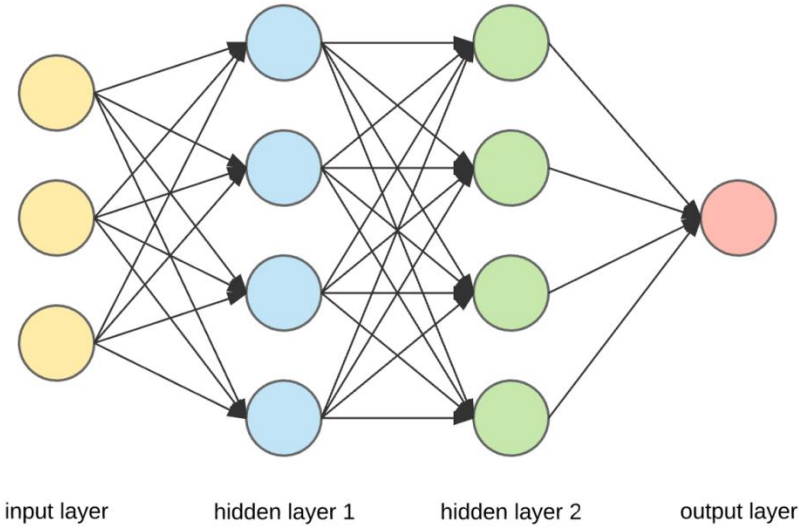

Figura 5: Red neuronal

Como podemos observar en la figura 5, las redes neuronales están compuestas por capas y estas a su vez están formadas por nodos que se conectan entre ellos. Estos nodos reciben también el nombre de **neuronas** y son indispensables para el funcionamiento de la red neuronal. Cada una de las capas tiene su propia función:

- **Capa de entrada (*input layer*):** Se trata de la entrada de la red neuronal, estos nodos proporcionan información del mundo exterior a la red. Las neuronas de esta capa no realizan ningún tipo de computación, solo pasan la información a las capas ocultas.
- **Capa oculta (*hidden layer*):** Los nodos de la capa oculta no tienen una conexión directa con el mundo exterior, por eso la capa se llama "oculta". Ellos se encargan de realizar cálculos y transferir la información que reciben desde la capa de entrada a la capa de salida.
- **Capa de salida (*output layer*):** Los nodos de la capa de salida son los responsables de realizar cómputos y de transferir la información de la red neuronal al mundo exterior.

Una red neuronal siempre debe tener una capa de entrada y otra de salida. Sin embargo, puede tener desde cero a múltiples capas ocultas.

Neurona

Es la unidad básica de computación en una red neuronal. Una neurona recibe información de otras neuronas o de una fuente externa y calcula un *output*. Cada input tiene asociado un **peso (weight (w))**, el cual indica la importancia que tiene el propio *input* respecto a los otros que recibe. La neurona aplica una **función f** a la suma de los pesos de sus entradas y genera una salida como podremos ver en la siguiente figura.

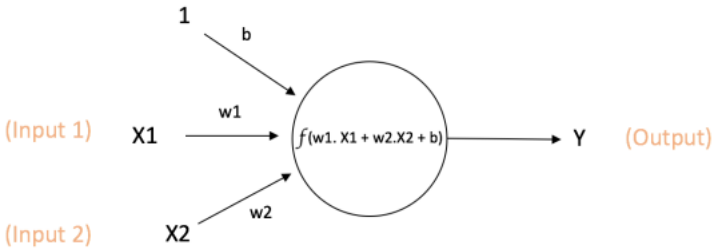

Figura 6: Funcionamiento de una neurona

Función f

Se trata de una función no lineal y es llamada **Función de Activación**. Su propósito es la de introducir no linealidad a la salida de una neurona. Esto es importante ya que los datos del mundo real son no lineales. Por lo que queremos que nuestras neuronas aprendan de estas representaciones no lineales.

Cada función de activación coge un número y realiza una operación matemática sobre él. Las funciones de activación que se utilizan en este proyecto son:

- **ReLU (*Rectified Linear Unit*):** Esta función de activación recibe una entrada y en caso de ser un valor negativo lo convierte en cero.

$$f(x) = max(0, x)$$

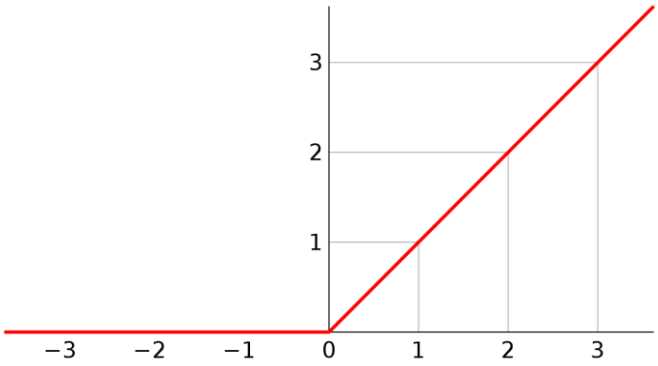

Figura 7: Función ReLU

- **Lineal:** Se trata de una función que corresponde a una línea recta con **μ** siendo la pendiente.

$$f(x) = \mu x$$

- **Softmax:** La función softmax puede ser utilizada para representar una distribución de probabilidad sobre **k** diferentes posibles salidas.

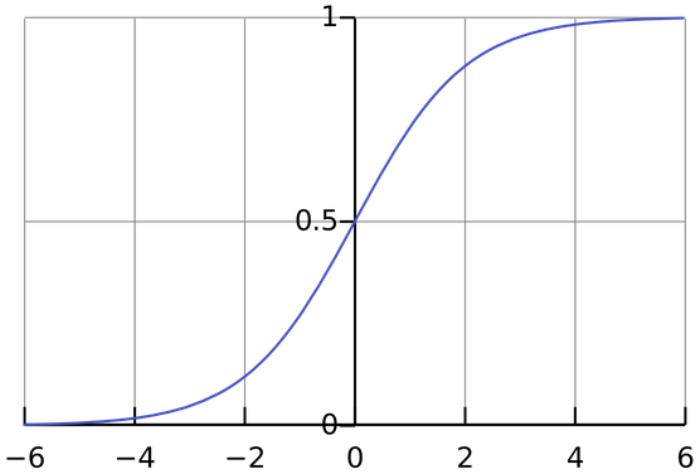

Figura 8: Función Softmax

4.2. Algoritmos

En este apartado veremos los algoritmos de aprendizaje por refuerzo que se han utilizado en este proyecto para entrenar a los agentes.

4.2.1. Q-Learning

Q-Learning [12] [13] [14] es un algoritmo de aprendizaje por refuerzo *off-policy* que tiene como objetivo maximizar el Q-value. Esto significa que para cada estado buscará cual es la mejor acción a realizar utilizando una política **greedy** o **voraz**.

El algoritmo básico de Q-learning utiliza una tabla llamada Q-table para guardar los valores Q(s, a). Sin embargo, en este proyecto se utilizará una red neuronal para almacenar dichos valores. Q(s, a) representa lo bueno que es realizar la acción a en el estado s o más formalmente, como se ha explicado anteriormente, se trataría del refuerzo esperado a largo plazo con descuento al realizar la acción a en el estado s.

El algoritmo de Q-learning es bastante sencillo ya que consiste en la actualización de los *Q-values* de forma iterativa.

1. Initialize Q-values ($Q(s, a)$) arbitrarily for all state-action pairs.
2. For life or until learning is stopped...
3. Choose an action (a) in the current world state (s) based on current Q-value estimates ($Q(s, \cdot)$).
4. Take the action (a) and observe the the outcome state (s') and reward (r).
5. Update $Q(s, a) := Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$

Figura 9: Pseudocódigo Q-learning

Para empezar, todos los *Q-values* de cada estado y para cada acción se inicializan de forma arbitraria. A continuación, cada vez que el agente selecciona una acción a, recibe una recompensa inmediata r y un nuevo estado s'. Finalmente, actualiza el valor Q(s, a) utilizando la siguiente formula:

$$NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma max Q'(s',a') - Q(s,a)]$$

- $NewQ(s,a)$: New Q value for that state and that action
- $Q(s,a)$: Current Q value
- α: Learning Rate
- $R(s,a)$: Reward for taking that action at that state
- γ: Discount rate
- $max Q'(s',a')$: Maximum expected future reward **given the new s' and all possible actions at that new state**

- **Ratio de aprendizaje/*Learning rate* (α):** Es un valor perteneciente al intervalo [0, 1] que indica cuanto aprendemos de la nueva experiencia. Si fuera 0, no aprenderíamos nada, mientras que si fuera 1 olvidaríamos todo y nos quedaríamos solo con los valores de la nueva experiencia.
- **Factor de descuento/*Discount rate* (γ):** Se trata también de un valor entre 0 y 1 que indica la importancia que le damos a la recompensa inmediata o a la recompensa a largo plazo. Si el valor fuera 0 solo le daríamos importancia a la recompensa inmediata, mientras que por otro lado, al ser 1 le daríamos solo importancia a la recompensa futura.

4.2.2. SARSA

SARSA [14] es muy parecido a Q-learning. Sin embargo, la diferencia clave entre un algoritmo y otro es que SARSA es un algoritmo *on-policy*, es decir, SARSA aprende los Q-*values* basados en las acciones realizadas por la política actual y no por la política greedy como hace Q-learning.

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

Figura 10: Pseudocódigo SARSA

En la figura 11 podemos observar lo parecido que es SARSA a Q-Learning. La diferencia es que al calcular los Q-*values* Q-learning mira cual es el argumento máximo en s', en cambio SARSA realiza una acción a' en s' y obtenemos su recompensa. Esto se puede observar también en la fórmula de actualización de los Q-*values* a continuación:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

En la formula se puede ver claramente como SARSA utiliza el Q-*value* del estado s_{t+1}, con la nueva acción elegida a_{t+1} por tal de actualizar el valor actual.

4.2.3. Actor-Critic (A2C)

El algoritmo A2C [15] es totalmente diferente a Q-learning y SARSA ya que utiliza dos redes neuronales. Cada una de ellas tiene una función y son las que dan nombre a este algoritmo.

- ***Actor*:** Es la red que controla el comportamiento del agente, es decir, controla la política.
- ***Critic*:** Se trata de la red que mide lo buena que es una acción que se ha realizado. Esto lo sabrá teniendo en cuenta la recompensa que se ha obtenido al realizar esa acción.

Para visualizar mejor este funcionamiento, imaginemos que una persona está jugando con un amigo a un juego y este a su vez le proporciona *feedback* de como juega. La persona sería el *Actor* y el amigo el *Critic*.

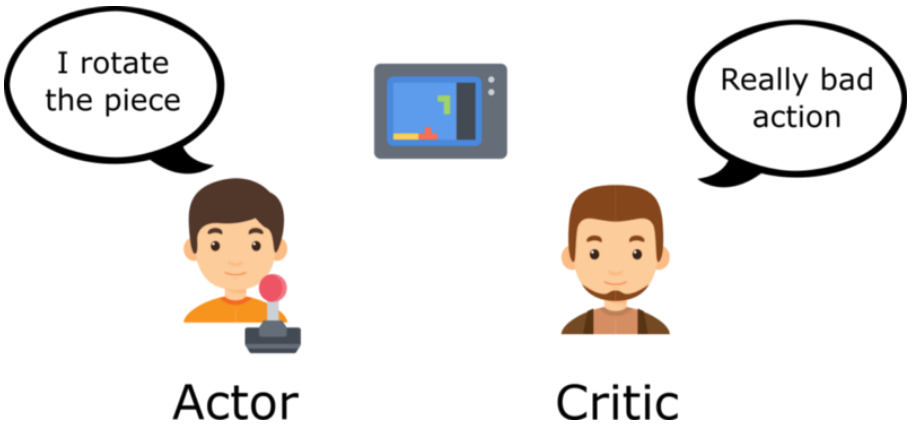

Figura 11: Como funciona Actor-Critic

Al principio como la persona no sabe jugar al juego, intenta hacer acciones de forma arbitraria. Entonces su amigo observa las acciones y le proporciona *feedback.*

Aprendiendo de este feedback, la persona cambiara su forma de jugar (**actualizara su politíca**) y jugará mejor. Por otro lado, su amigo también aprenderá como es la mejor forma de darle a la persona *feedback* por lo que dará un mejor *feedback* la próxima vez.

De esta forma estas dos personas funcionan igual que lo harán las dos redes neuronales en el algoritmo y estimamos ambas:

- **Función Actor:**

$$\pi(s, a, \theta)$$

- **Función Critic:**

$$\hat{q}(s, a, w)$$

Ambas se ejecutan en paralelo y al ser redes diferentes significa que tendremos dos conjuntos de pesos (θ for our *Actor* and w for our *Critic*) que deben ser optimizados por separado.

El proceso del algoritmo es el siguiente:

Primero, en cada instante t, obtenemos el estado **St** del entorno y se lo enviamos como entrada al *Actor* y al *Critic*.

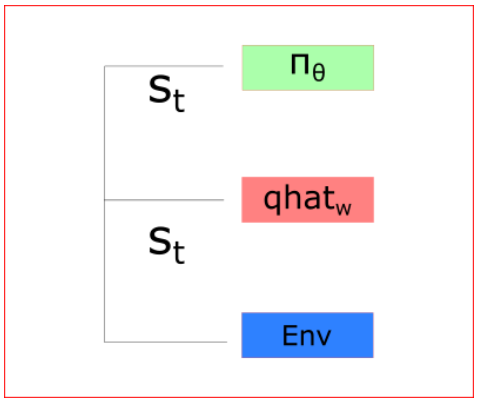

Figura 12: Proceso Actor-Critic 1

Nuestra política coge el estado, como salida envía una acción **At** y se obtiene un nuevo esta **St+1** y una recompensa **Rt+1**. Esto hace que el *Critic* calcule el valor de realizar esa acción en ese estado y que el *Actor* actualicé sus parámetros de política usando el Q-value resultante.

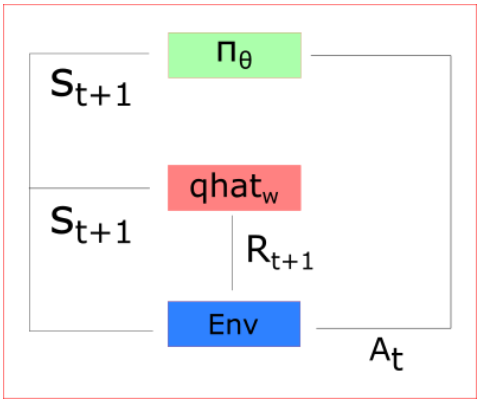

Figura 13: Proceso Actor-Critic 2

La fórmula de actualización del *Actor* es la siguiente:

$$\Delta\theta = \alpha \, \nabla_\theta(log \, \pi_\theta(s,a))\hat{q}_w(s,a)$$

Gracias a sus parámetros actualizados, el *Actor* genera la siguiente acción a realizar **At+1** dado un nuevo estado **St+1**. En este momento el *Critic* actualiza sus parámetros *value.*

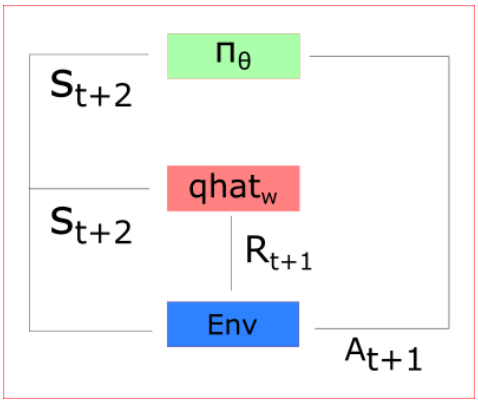

Figura 14: Proceso Actor-Critic 3

Las fórmulas de actualización del *Critic* es la siguiente:

$$\Delta w = \beta \, (R(s,a) + \gamma\hat{q}_w(s_{t+1}, a_{t+1}) - \hat{q}_w(s_t, a_t))\nabla_w\hat{q}_w(s_t, a_t)$$

4.3. Experience Replay

Se trata de una técnica que utilizarán los algoritmos de este proyecto. Mientras se está produciendo el entrenamiento, todas las experiencias (<s, a, r, s'>) son almacenadas en una memoria llamada *replay memory*. A la hora de entrenar la red neuronal, se obtienen muestras de forma arbitraria de la *replay memory* y se usaran en vez de la transición más reciente. Esto rompe la similitud entre muestras subsiguientes de entrenamiento por tal de evitar que la red neuronal pueda acabar en un mínimo local. [12]

5. Escenarios de Entrenamiento

En este apartado se describirán en detalle los escenarios de entrenamiento que se han utilizado durante el proyecto. ¿Qué retos les depararán a nuestro jugador y a nuestro agente aliado.

5.1. Harvest Wood

Un día nuestro jugador necesita madera para construirse una pequeña casa cerca de la orilla del rio. Nuestro agente muy amablemente le dijo que le ayudaría a recoger madera para construirla. ¿Serán capaces de recoger toda la madera?

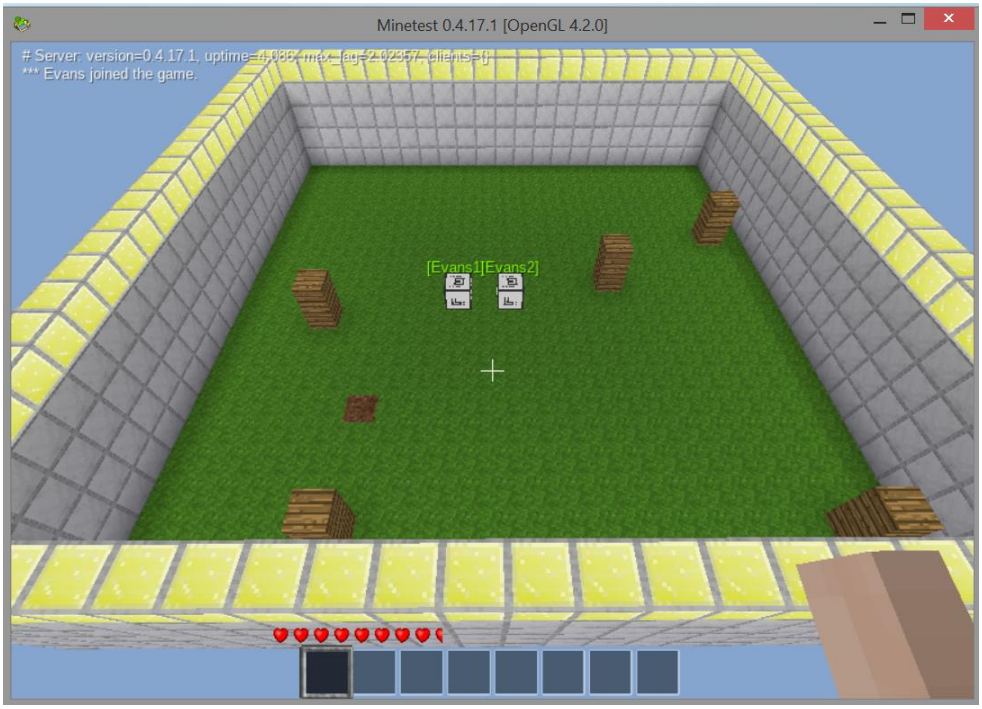

Figura 15: Escenario de entrenamiento Harvest Wood en Minetest

5.1.1. Descripción y objetivos

El objetivo de este escenario es recolectar tantos cubos de madera como sea posible en un plazo de 500 turnos. En el momento que se recolecten todos los cubos de madera del escenario también finalizaría la iteración del entrenamiento.

5.1.2. Elementos a tener en cuenta

En el escenario hay 5 pilas de madera, por lo que habrá un total de 10 cubos de madera a recoger. Los cubos de madera formaran pilas de 2 cubos cada una y se generarán en el escenario de forma aleatoria. Cabe decir que el jugador ya tendrá 1 cubo de madera en el inventario para que el agente vea de su interés en ese material. En conclusión, el total de cubos de madera en este escenario de entrenamiento será de 11.

El agente debe comportarse como un aliado que ayuda al jugador, así que también se medirá la distancia que hay entre el jugador y el agente para ver si realmente lo acompaña o se marcha solo sin tenerlo en cuenta.

5.1.3. Comportamiento del jugador

En este escenario el jugador quiere que el agente recolecte madera y a la vez lo siga, lo que para facilitar ese comportamiento el jugador se aproximará a cada una de las pilas de madera y escogiendo a cual se moverá de forma aleatoria. Es decir, no se moverá primero a por la pila más cercana sino que puede ser cualquiera de ellas. De esta forma, si el agente le sigue lo suficientemente cerca, dará con las pilas de madera fácilmente.

5.1.4. Recompensas

Con el objetivo de recolectar madera y no mantenerse a una distancia muy lejana al jugador las recompensas son:

- El agente se mueve a una posición que se encuentra a una distancia de 6 posiciones o inferior respecto al jugador = **2 punto.**
- Si el agente gira a izquierda o derecha debido a que ha detectado que hay una pila de madera a uno de sus lados = **1 punto.**
- Si el agente salta sobre un cubo y desde allí dispone de una mejor posición para coger un cubo de madera = **2 puntos.**
- El agente recolecta un cubo de madera. La recompensa dependerá del número de cubos de madera que ya tenga en el inventario para incentivar que recoja todos los posibles:
    - 1er cubo = **1000 puntos.**
    - 2ndo cubo = **2000 puntos.**
    - 3er cubo = **3000 puntos.**
    - 4rto cubo = **4000 puntos.**
    - …
    - 10mo cubo = **10000 puntos.**

En caso de que deje un cubo en el suelo o se lo entregue al jugador, de los que dispone en el inventario hasta el momento, volverá a empezar por la recompensa de recoger el 1er cubo en el caso de la entrega o volverá a la recompensa anterior en caso de dejar un cubo en el suelo.

- El agente entrega cubos de madera al jugador. Ya no solo hay que valorar todo lo que recoge el agente, sino que como un buen aliado debe ayudar al jugador con su tarea. Así que esta acción será recompensada = **250 puntos por cubo de madera.**
- Penalización de **-1 punto** por acciones no validas como intentar moverse a una posición fuera del escenario.
- El agente recolecta todos los cubos de madera del escenario = **100000 puntos.**

5.2. Chase the Player in the Labyrinth

Después de recoger la madera que pudieron, el agente y jugador volvían a casa cansados. Nuestro jugador aseguró al agente saber un atajo para llegar antes, sin embargo, acabaron metidos en un laberinto. Las palabras literales del jugador fueron "No recordaba que estuviera ahí". ¿Conseguirán salir de esta?

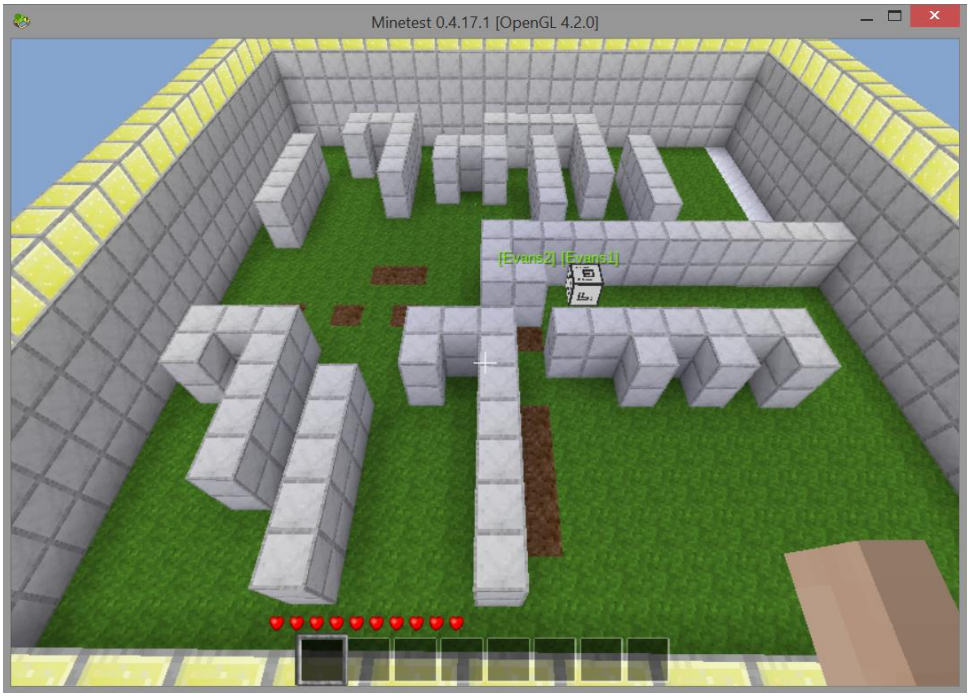

Figura 16: Escenario de entrenamiento Chase the Player in the Labyrinth en Minetest

5.2.1. Descripción y objetivos

El objetivo de este escenario es seguir al jugador por tal de poder salir del laberinto en un plazo de 500 turnos. En el momento que el agente llegue a una de las casillas de salida también finalizaría la iteración del entrenamiento.

5.2.2. Elementos a tener en cuenta

Las casillas de salida son las que deberán alcanzar para poder superar el entrenamiento y se elegirá de forma aleatoria en cuál de los 3 extremos del escenario, diferentes del más cercano a la posición inicial del agente y del jugador, se generará.

El laberinto se genera totalmente de forma aleatoria a partir de 3 tipos de piezas, cada una con una forma diferente y formada por pilas de 3 cubos de acero. También se generará una pared, que no cambiará a lo largo de las iteraciones, cerca de la posición del jugador y el agente para que no puedan evitar pasar por el laberinto en el momento de buscar la salida.

El agente como buen aliado debe seguir al jugador y salir junto a él del laberinto. Así que en este escenario se medirá a las diferentes distancias que se encuentra en cada turno respecto al jugador.

5.2.3. Comportamiento del jugador

En este escenario el jugador sabrá en qué lugar está la salida más cercana. Seguirá el camino más corto por el laberinto hasta llegar a dicha posición esperando que el agente le siga.

5.2.4. Recompensas

Con el objetivo de que el agente siga al jugador y no se mantenga a una distancia muy lejana de él las recompensas son:

- El agente se mueve a una posición que se encuentra a una distancia de 1 o 2 posiciones respecto al jugador = **1000 puntos,**
- El agente se mueve a una posición que se encuentra a una distancia de 3 o 4 posiciones respecto al jugador = **50 puntos.**
- El agente se mueve a una posición que se encuentra a una distancia de 5 posiciones respecto al jugador = **10 puntos.**
- Si el agente se encuentra atrapado en el laberinto porque no puede avanzar más y decide girar a derecha o izquierda por tal de darse la vuelta y seguir por otro camino = **1 punto.**
- Penalización de **-1 punto** por acciones no validas como intentar moverse a una posición fuera del escenario.
- El agente llega a una casilla de salida y consigue salir del laberinto = **100000 puntos.**

5.3. Survive the Deadly Road

Tras salir del laberinto, nuestro jugador y agente se pensaban que no volverían a encontrarse con ninguna dificultad de vuelta a casa. Se equivocaban… El volcán del pueblo de al lado de la ciudad del jugador entró en erupción en el tiempo que estuvieron fuera y lleno de lava una de las rutas que tenían que tomar. Por suerte algún vecino del pueblo puso un camino para atravesar el mar de lava que se encontraba ante ellos. ¿Lograrán cruzarlo?

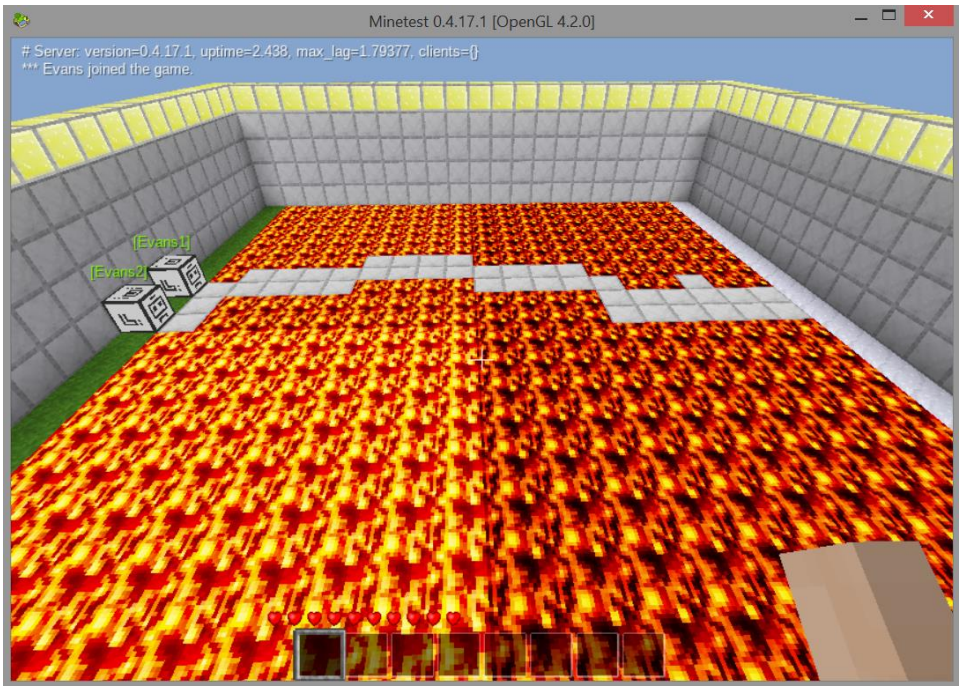

Figura 17: Escenario de entrenamiento Survive the Deadly Road en Minetest

5.3.1. Descripción y objetivos

El objetivo de este escenario es el de seguir al jugador por tal de cruzar el mar de lava utilizando un pequeño camino de acero que lo atraviesa. El plazo para conseguirlo será de 500 turnos y en esta ocasión la iteración puede finalizar de dos formas más:

1. El agente consigue cruzar el camino y se completa la iteración satisfactoriamente.
2. El agente se cae a la lava y muere. Por lo que la iteración finalizara de forma fallida.

5.3.2. Elementos a tener en cuenta

El camino que cruza el mar de lava se generará a partir de 4 tipos de piezas que se elegirán de forma aleatoria por tal de construirlo desde la posición donde se encuentran el jugador y el agente hasta la otra orilla del mar de lava. El camino no dispone de protecciones, así que un error y el agente fracasará en la iteración.

Las casillas objetivo siempre se encontrarán en la otra orilla del mar de lava y no cambiaran.

El agente como buen aliado debe seguir al jugador y no caer en la lava. En este escenario se medirá la distancia respecto al jugador y que el agente se mantenga en el camino.

5.3.3. Comportamiento del jugador

En este escenario el jugador sabrá cuál es la mejor forma de cruzar el mar de lava. Seguirá el recorrido más corto por el camino de acero hasta llegar a cruzar totalmente el mar y esperará al agente en la otra orilla.

5.3.4. Recompensas

Con el objetivo de que el agente no se mantenga a una distancia muy lejana del jugador y tampoco se caiga a la lava las recompensas son:

- El agente se mueve a una posición en la que se mantiene en el camino de acero y se encuentra a una distancia de 5 posiciones o inferior respecto al jugador = **1000 puntos.**
- El agente se mueve a una posición en la que se mantiene en el camino de acero y se encuentra a una distancia mayor a 5 posiciones respecto al jugador **= 100 puntos.**
- Si el agente detecta que si avanza se va a caer a la lava y decide girar a izquierda o derecha = **1 punto.**
- Penalización de **-100 puntos** si el agente cae a la lava y es eliminado.
- El agente cruza el mar de lava y llega a la otra orilla = **100000 puntos.**

5.4. Fight Zombies

Una vez nuestro jugador y nuestro agente llegaron a la villa del jugador tras todos los percances del camino, se encontraron con que había estallado el apocalipsis zombi en el mundo y habían destruido la querida casa de nuestro jugador. El jugador habiéndose quedado sin casa juró venganza y el agente juró que le defendería y acabaría con todos. ¿Conseguirán eliminar a todos los zombis?

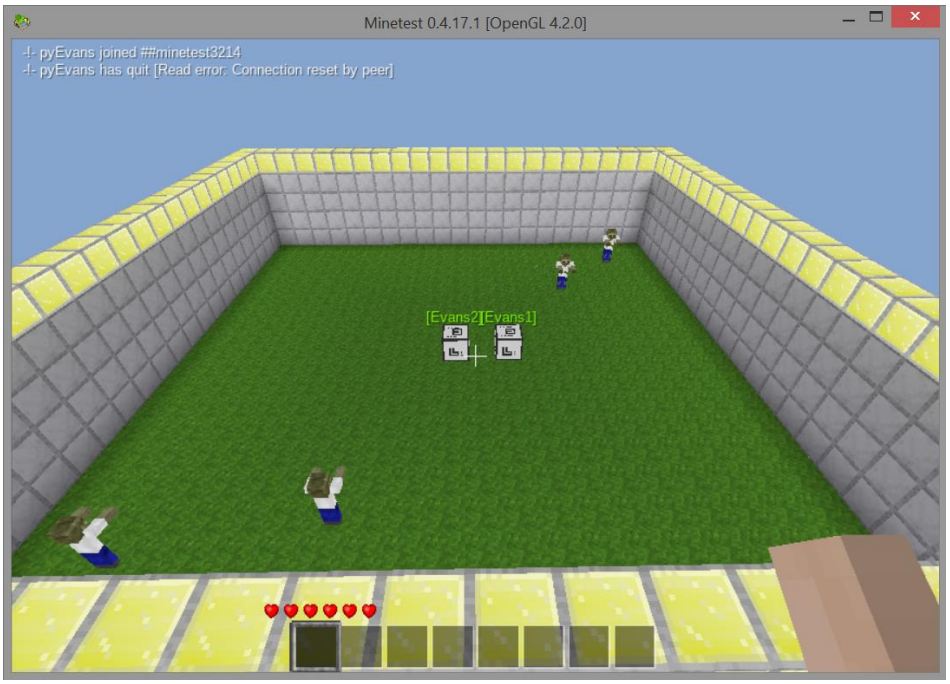

Figura 18: Escenario entrenamiento Fight Zombies en Minetest

5.4.1. Descripción y objetivos

El objetivo de este escenario es el de eliminar tantos zombis como sea posibles en un plazo de 500 turnos. En el momento que todos los zombis hayan sido eliminados terminará la iteración del entrenamiento.

5.4.2. Elementos a tener en cuenta

En este escenario habrá 4 zombis y aparecerán aleatoriamente 2 en la parte superior del escenario y 2 en la parte inferior del escenario. Se ha tenido en cuenta una distancia mínima para que no reaparezcan justo al lado del jugador y el agente.

Por tal de fomentar que el agente elimine zombis, el jugador solo podrá dañar a los zombis, pero no eliminarlos. Si cada zombi necesita recibir 2 ataques para ser eliminado, el jugador solo puede realizar 1 ataque y el necesario para la eliminación tendrá que ser por parte del agente.

La distancia a partir de la cual el jugador o el agente pueden atacar a los zombis o viceversa lo llamaremos rango de ataque. El rango de ataque del jugador y del agente será de 3, mientras que los zombis solo dispondrán de un rango de ataque de 1.

Los zombis no son muy inteligentes, pero todos sabemos que pueden escuchar muy bien lo que hay cerca por tal de atacar a una presa. Los zombis disponen de lo que hemos llamado rango de búsqueda que detectará al jugador si se encuentra dentro de este rango y le seguirán para atacarle. El rango de búsqueda de los zombis es de 4.

Los zombis realizan movimientos y actúan, por lo que no serán objetivos estáticos. Se ha planteado en este escenario que los zombis se muevan y ataquen solo al jugador por tal de que el agente intente eliminarlos antes de que dañen al jugador y protegerle o simplemente cuando haya recibido el mínimo daño posible.

En este escenario se medirá la distancia al jugador y el daño que tanto jugador como zombis reciban.

5.4.3. Comportamiento del jugador

En este escenario el jugador sabe cuál es la posición de cada zombi. Se moverá desde el centro del escenario a los alrededores por tal de huir de los zombis. Sabe dónde se encuentra el zombi más cercano y si se encuentra dentro del rango de ataque le dañará y esperará que el agente lo elimine.

5.4.4. Comportamiento de los zombis

Los zombis se moverán después de que el jugador y el agente hayan hecho su movimiento en un turno. Ellos son unos seres violentos, así que lo primero que harán es atacar al jugador si este se encuentra dentro del rango de ataque. En caso de que no fuera así, intentarán escuchar para ver si el jugador se encuentra dentro del rango de búsqueda y entonces moverse hacia su posición.

Si el jugador no se encuentra ni dentro del rango de ataque ni del de búsqueda de un zombi entonces este hará gala de su famosa inteligencia. Se moverá aleatoriamente o simplemente se quedará atontado sin moverse de la posición en la que se encontraba.

5.4.5. Recompensas

Con el objetivo de que el agente no se mantenga en una posición muy lejana del jugador y elimine a los zombis las recompensas son:

- Si el agente se mueve a una posición que se encuentra a una distancia de 6 posiciones o menos respecto al jugador = **2 puntos.**
- El agente gira a izquierda o derecha para encontrar una posición que se encuentra a una distancia de 6 posiciones o menos respecto al jugador = **1 punto.**
- Si el agente ataca a un zombi y este continúa vivo = **100 puntos.**
- El agente ataca a un zombi y lo elimina = **500 puntos.**
- Penalización de **-1 punto** por acciones no validas como intentar moverse a una posición fuera del escenario.
- Penalización de **-100 puntos** si el jugador recibe daño por parte de un zombi.
- Todos los zombis son eliminados = **5000 puntos.**

6. Definición del Estado y Acciones

La definición del estado y las acciones es una de las partes más importantes del proyecto debido a la relevancia que tendrá sobre los resultados finales.

Si tuviéramos en cuenta cada elemento que hay en un mapa del juego de Minetest para que el agente tuviera la información correspondiente a cada espacio de un bloque, cuando un mundo completo tiene más de 200 cuatrillones de nodos, resultaría de un espacio de estados increíblemente enorme. Imposible para los recursos disponibles en este proyecto y quién sabe lo desafiante que sería para los supercomputadores que tienen disponibles Google o Microsoft.

Un jugador de Minetest tiene diversas acciones desde caminar más rápido e ir de puntillas hasta las diferentes selecciones de cámaras o de que objeto del inventario llevar equipado. Sin embargo, no todas ellas serían útiles para el entrenamiento del agente y añadirlas solo harían que ocupar una gran parte en el espacio de acciones.

Pensando en querer obtener los mejores resultados para el proyecto, se han elegido las características del entorno para el estado y las acciones para el agente que hemos creído mejores para reducir el espacio de estados y de acciones para el entrenamiento. Quizá hubiera mejores o diferentes opciones, pero las elegidas en este proyecto se explicaran a continuación.

6.1. Estado

Las características del mundo de Minetest que se han tenido en cuenta en este proyecto para el estado son las siguientes:

- **Nodos/Casillas de la posición del agente aliado y sus alrededores:** Es la parte más importante del estado y la que dispondrá de más elementos en el mismo. Será imprescindible para el agente ya que es lo que le permitirá situarse en el mundo de Minetest y saber que cubos tiene exactamente a su alrededor, encima de él o incluso debajo. Se explicarán más detalles en el siguiente apartado.
- **Posición del jugador:** Queremos que el agente sea un aliado del jugador por lo que será muy importante que sepa su posición y le acompañe o ayude de la mejor forma posible teniéndola en cuenta y no dejarle completamente solo. La posición del jugador constará de 3 elementos, las coordenadas x, y, z de su posición.

- **Posición del enemigo más cercano:** Este apartado será de utilidad en el momento que el agente entrene en el escenario *Fight Zombies*. De esta forma tendrá en cuenta donde se encuentra el zombi más cercano respecto a su posición. En cualquier otro escenario esta parte no afectara al estado. La posición del enemigo más cercano constará de 3 elementos, las coordenadas x, y, z de su posición.
- **Inventario del jugador:** Los inventarios forman un papel imprescindible en el escenario *Harvest Wood*. El agente debe saber que materiales tiene en el inventario el jugador por tal de buscarlos ya que serán los que les interesen. El inventario del jugador constará de 8 elementos, correspondientes al inventario principal del que el jugador dispone en una partida de Minetest.
- **Inventario del agente aliado:** Igual de importante que el inventario del jugador para *Harvest Wood.* El agente debe saber de qué cubos dispone en su propio inventario tanto para colocar uno de los cubos en el escenario o darle cubos de materiales que le interesan al jugador. El inventario del agente constará de 8 elementos, correspondientes al inventario principal del que el jugador dispone en una partida de Minetest.

Detalles de los nodos/casillas de la posición del agente aliado y sus alrededores

Debido a su importancia me gustaría entrar en más detalle en cual es cada elemento y como se ha llegado a escoger estos distintos elementos.

En una posición, el agente obtendrá información de 4 cubos/nodos:

- Nodo que se encuentra 1 posición debajo de la altura del agente.
- Nodo que se encuentra en la misma posición que la altura del agente.
- Nodo que se encuentra 1 posición encima de la altura del agente.
- Nodo que se encuentra 2 posiciones encima de la altura del agente.

De esta forma el agente puede ver en una posición que hay justo debajo, enfrente y encima teniendo en cuenta su altura.

Por tal de obtener un mejor comportamiento se han realizado unas breves pruebas con 3 configuraciones de estos elementos:

- **Configuración 1:**

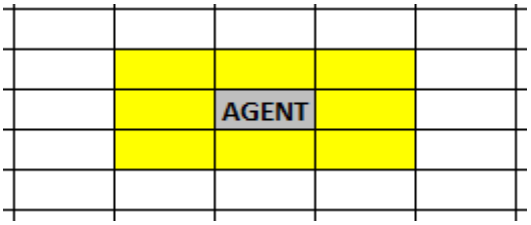

Figura 19: Configuración 1 - posiciones que tiene en cuenta el agente

Como podemos observar en la figura 18, con esta configuración el agente obtiene información de los 4 nodos, como se ha explicado anteriormente, tanto de la posición en la que se encuentra como de las posiciones adyacentes.

Esta configuración constaría de 36 elementos para el estado.

- **Configuración 2:**

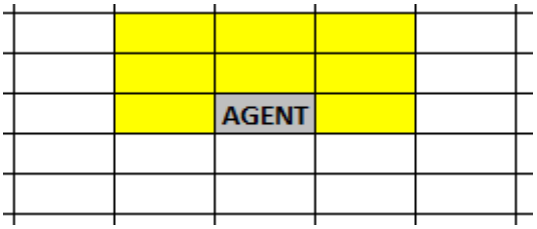

Figura 20: Configuración 2 - posiciones que tiene en cuenta el agente

La configuración 2 es parecida a la 1, pero sacrificando las posiciones que el agente tiene a sus espaldas para acabar añadiéndole más vista frontal.

Esta configuración también constaría de 36 elementos para el estado.

- **Configuración 3:**

Figura 21: Configuración 3 - posiciones que tiene en cuenta el agente

Se trata de la combinación de la configuración 1 y 2. Tiene tanto visión posterior como mayor alcance de la vista frontal.

Esta configuración constaría de 70 elementos para el estado, superior en número a las dos anteriores.

Para tomar esta decisión las pruebas se realizaron en la etapa del comienzo del entrenamiento en el escenario *Harvest Wood*:

Configuración	Máxima Recompensa	Recompensa Media
1	6782	515
2	6786	519
3	4519	371

Tabla 12: Resultados de las pruebas de las configuraciones del estado

Como podemos observar la configuración 3 es la que ha obtenido una menor puntuación de las tres. Además, sería la configuración que ofrecería un espacio de estados superior a las otras dos. Acabó descartada.

Respecto a las otras dos, ambas obtienen una puntuación parecida, así que se ha tomado otro factor en cuenta para tomar la elección: cómo ve el mundo un jugador de Minetest. Un jugador de Minetest dispone de una visión mayor que solo los cubos que tiene 1 posición hacia delante y no puede saber qué es lo que hay justo detrás de él.

Finalmente, se acabó eligiendo la configuración 2 para el estado sobre las otras dos.

Estado Final

El estado finalmente dispondrá de 58 elementos conteniendo toda la información mencionada anteriormente.

6.2. Acciones

En este apartado definiremos las acciones que tiene a su disponibilidad el agente por tal de asemejar su comportamiento a un jugador de Minetest.

- **MOVE:** Se trata de la acción básica de movimiento. El mundo de Minetest está dividido en nodos/cubos en los tres ejes de coordenadas (x, y, z) por lo que significará un cambio de posición desde la actual a la deseada. Esta acción permite al agente moverse en distintas direcciones:
    - **Forward:** Se mueve una posición hacía delante.
    - **Backward:** Se mueve una posición hacía atrás.
    - **Left:** Se mueve una posición hacía la izquierda (sin cambiar su punto de vista).
    - **Right:** Se mueve una posición hacía la derecha (sin cambiar su punto de vista).

    - **Jump:** Se trata de la opción de dirección más compleja. Con ella el agente puede saltar sobre bloques para superar obstáculos o recolectar materiales que no podía anteriormente desde una altura inferior. El salto se producirá siempre y cuando no se encuentre ningún obstáculo por medio.

  Destacar que cualquiera de estas acciones si se realizan desde una altura superior a la del suelo predeterminado de Minetest, se les aplicará el efecto de la gravedad en el caso de que no tengan ningún nodo debajo al realizar el movimiento.

- **TURN:** Es la acción que permite rotar 90 grados al agente por tal de cambiar la dirección en la que "ve". Esta acción dispone de dos opciones:
    - **Right:** Realiza un giro hacia la derecha.
    - **Left:** Realiza un giro hacia la izquierda.

- **DIG:** Se trata de la acción que utilizará el agente por tal de obtener materiales del escenario. Puede obtenerlos desde diferentes direcciones:
    - **Forward:** Obtiene el material que está justo en frente del agente.
    - **Forward Up:** Obtiene el material que se encuentra a 1 más de altura delante de su posición.
    - **Forward Down:** Obtiene el material que se encuentra a 1 menos de altura delante de su posición.
    - **Up:** Obtiene el material que se encuentra justo encima del agente.
    - **Down:** Obtiene el material que se encuentra justo debajo del agente.

- **PLACE:** Es la acción que el agente utilizará para poner cubos, que previamente a recogido con la acción *dig*, desde su inventario en el escenario. En caso de que no disponga de ningún cubo en el inventario no se produciría ningún cambio. También puede colocar cubos en diferentes direcciones:
    - **Forward:** Coloca un cubo delante de él.
    - **Forward Up:** Coloca un cubo a 1 más de altura delante de su posición. No puede colocar nada en el caso de que no haya ningún material delante de él para poner el cubo encima.
    - **Forward Down:** Coloca un cubo a 1 menos de altura delante de su posición. No puede colocar nada en el caso de que no haya ningún material delante de él para poner el cubo debajo. También serviría ponerlo sobre el suelo.

- **GIVEDROPS:** Utilizando esta acción el agente le dará al jugador cubos de materiales. Se le indica un tipo de material y si el agente lo tiene en el inventario se lo dará al jugador. En caso de que no tenga ese tipo de material no se producirá ningún efecto.

- **ATTACK:** El agente parecía inofensivo, pero no es así. Puede atacar y no dudará en usar esta acción ante sus enemigos.

7. Implementación

En términos de implementación del proyecto hay 2 grandes apartados a tener en cuenta y los trataremos a continuación.

7.1. Implementación dentro de Minetest

Esta parte de la implementación se ha realizado trabajando en scripts que se encuentran dentro del juego de Minetest. Para ello se han utilizado los lenguajes de programación de Python y Lua.

7.1.1. Conexión de Minetest

Para establecer la conexión con Minetest se ha utilizado un *mod* del juego junto a su librería de Python llamado **ircbuilder**. Esta herramienta hace como intermediario entre el Lua de Minetest y código programado desde fuera del videojuego y permite el envío de información entre ellos.

Su funcionamiento se basa en establecer una conexión con el chat de juego del servidor de un mundo de Minetest y enviar comandos, programados anteriormente, que al recibirlo el chat del juego lo ejecutara y producirá cambios en el mundo de Minetest.

7.1.2. Personajes del jugador, agente aliado y zombis

Por tal de observar el comportamiento de los entrenamientos tanto del jugador que hemos programado como del agente en Minetest necesitábamos de un modelo para los personajes. Para ello se ha utilizado el *mod* **basic_robot** de Minetest. Esta herramienta está pensada para que cualquier persona que quiera aprender el lenguaje de programación Lua pueda hacerlo de una forma divertida utilizando Minetest.

El *mod* permite programar un robot en Lua y dispone de los elementos necesarios para ello: un modelo, un *spawn* para el robot y funciones básicas de movimiento, codificadas dentro de Minetest, y spawn. Un jugador iría al *spawn* que ha colocado dentro del juego y se le abriría una terminal donde podría introducir su código. Sin embargo, este no es el uso que le daremos en este proyecto. Nosotros utilizaremos los modelos del robot para los personajes del robot y agente y utilizaremos la función de *spawn*, pero no literalmente el resto.

En nuestro proyecto funcionara de la siguiente forma: Primero, ejecutaremos el código fuera de Minetest y ese código hará llamadas a funciones definidas en la conexión, que hemos explicado en el apartado anterior, que cada una de estas funciones enviara un mensaje que activará un comando del chat de dentro de la partida y permitirá que el jugador y el agente realicen acciones y movimientos.

Las características que dispondrán el jugador y el agente dentro de Minetest son las siguientes:

- **Nombre/*Name*:** String del nombre con el que se reconocerá al jugador o al agente dentro de Minetest.
- **Posición:** Array de coordenadas del nodo del mundo de Minetest donde se encuentra un personaje. Será un array de Lua, pero su funcionamiento es como si fuera un diccionario en Python.
- **Posición del *spawn*:** Indica las coordenadas donde se encuentra el objeto que permite hacer el *spawn* de los modelos en el mundo de Minetest. Se trata también de un array.
- **Inventario:** Es donde el jugador y el agente guardaran los cubos de materiales. El inventario está asociado al objeto de *spawn*. El inventario será de tipo array.

Al añadir al jugador y el agente al mundo de Minetest sus datos se almacenaran en un array de Lua llamado **basic_robot.data**. Este array tendrá como llave el nombre del jugador o el agente y el contenido será el objeto con toda la información correspondiente a cada uno de ellos. Todas las funciones tendrán que obtener la información del jugador o del agente de este array.

Respecto a los zombis, se ha utilizado un *mod* llamado **cme** que permite añadir a Minetest criaturas malignas y animales. Hemos utilizado el script de zombis por tal de añadirlos, pero este código ya tenía implementado un comportamiento para ellos. Se modificó el código por tal de anular todos sus movimientos y poder quedarnos solo con el modelo del zombi por tal de que se mueva y actúe como deseamos.

Las características que dispondrán los zombis dentro de Minetest son las siguientes:

- **Nombre/*Name*:** El nombre con el que se reconocerá al zombi dentro de Minetest. Es de tipo string.
- **Posición:** Array de las coordenadas del nodo del mundo de Minetest donde se encuentra un zombi.
- **Vida/HP(*Health Points*):** Son los puntos de vida que dispone un zombi. Cuando llegan a 0 el zombi será eliminado. Se trata de un número entero.

Los zombis al añadirse al mundo de Minetest se almacena su información en un array de Lua llamado **mobs.** La llave será el nombre del zombi y el contenido será el objeto correspondiente al zombi con ese nombre y contendrá toda su información. Toda función que necesite saber información de alguno de los zombis tendrá que consultar este array.

7.1.3. Envío de mensajes al chat

Esta parte es la única de dentro de Minetest que se ha realizado en Python. Por tal de enviar mensajes se utiliza la función **send_cmd** que cuenta con un único parámetro que llamaremos mensaje**.** Esta función básicamente manda un string con los caracteres cmd indicando que se trata de un comando y lo concatena con el parámetro mensaje de la función que también será un string. De esta forma el chat verá que está recibiendo un mensaje que es un comando y ejecutara el comando correspondiente.

Estas funciones se podrán ejecutar desde nuestro código de Python fuera de Minetest por tal de enviar comandos añadiendo su respectivo nombre y la información adicional necesaria. Las podemos ver a continuación:

- **send_building:** Se trata de una función ya implementada en el ircbuilder. Envía un diccionario de cubos y nodos con su respectiva posición, creado por la función **build** que se explicará más tarde, por tal de añadirlos al mundo de Minetest.
- **move_robot:** Envía un string indicando el nombre de un personaje, ya sea el del jugador o el del agente, por tal de mover el modelo dentro del mundo de Minetest.
- **turn_robot:** Envía un string indicando el nombre del personaje y la dirección en la que debe girar el modelo.
- **dig_robot:** El string es enviado con la información del nombre del personaje y la dirección en la que debe intentar obtener un cubo de material.
- **place_robot:** Se envía un string con la información del nombre del personaje y la dirección donde debe colocar un cubo de material proveniente de su inventario cuyo nombre también será indicado.
- **givedrops_robot:** En el string que envía se añade de información el nombre del jugador que entregará materiales, el receptor de dicho materiales y el nombre del material específico que se va a entregar.
- **add_starter_mat_to_player:** Utilizado en el escenario *Harvest Wood*. Envía un string con el nombre del jugador por tal de añadirle a su personaje un cubo de madera al inventario.

- **clean_inventories:** Envía un mensaje por tal de eliminar todos los elementos de los inventarios de los personajes del jugador y del agente obtenidos en una iteración anterior. También utilizado para *Harvest Wood.*
- **spawn_zombies:** Envía un mensaje indicando que se debe realizar el spawn de zombis en el escenario de *Fight Zombies.*
- **clean_zombies:** Utilizado en el escenario Fight Zombies. Envía un mensaje para indicar que se deben eliminar todos los zombis que hay en el mundo de Minetest.
- **kill_zombie:** Se envía un mensaje con el nombre del zombi en específico del mundo de Minetest para elimnarlo.
- **move_zombie:** Envía un mensaje indicando el nombre del zombi que va a realizar un movimiento.
- **turn_zombie:** El mensaje que se envía india el nombre del zombi que va a girar y en qué dirección.
- **get_player_pos:** Se envía un mensaje con el nombre de uno de los personajes por tal de obtener la posición en la que se encuentra.
- **get_pos_in_dir:** Este mensaje es enviado por tal de obtener la posición que se encuentra en una dirección respecto al personaje. Se indican el nombre del personaje y la dirección en el mensaje.
- **get_node_minetest:** Este mensaje se envía junto a las coordenadas de un nodo por tal de obtener el nombre de su contenido.
- **get_state_codes:** Envía varios mensajes con el nombre del jugador, el agente y el escenario en que se encuentra y usando todas aquellas direcciones necesarias para el estado del agente. De esta forma obtendrá un código asociado a cada uno de los materiales en la dirección indicada.
- **get_player_inventory_item:** Se envía un mensaje indicando el nombre de un personaje y un número indicando un espacio del inventario por tal de saber su contenido.
- **get_player_inventory_item_count:** Se envía un mensaje indicando el nombre de un personaje y un número indicando un espacio del inventario por tal de saber el número de materiales acumulados.

Funciones útiles que ayudan en el trabajo de algunas de las funciones anteriores son:

- **build:** Se le indican coordenadas y el nombre de un material y esta información se almacena en un diccionario que será el que utilizará **send_building**.
- **minetest_get_state:** Esta función obtiene el estado que utilizará la red neuronal a partir de la información del mismo juego de Minetest.
- **world_to_minetest_coord:** Transforma coordenadas del mundo simulado a coordenadas del mundo de Minetest.

- **minetest_to_world_coord:** Transforma coordenadas del mundo de Minetest al mundo simulado.
- **codify_content:** Se le pasa como parámetro el nombre de un material y lo transforma a su codificación apta para la red Neuronal.

7.1.4. Comandos y funciones

Esta parte ha sido totalmente implementada utilizando el lenguaje de programación Lua. Una vez la conexión de Minetest manda un mensaje al chat de juego y este detecta que se trata de un comando buscará en su lista de comandos si dispone de alguno que coincida con el nombre indicado en el mensaje. Es decir, para utilizar una función debe haberse creado antes un comando relacionado con el chat del juego que realice una llamada a ella. En caso contrario no existiría.

Las funciones que se han implementado para ejecutarse dentro de Minetest a través de los comandos del chat son:

- **move_robot:** Función que recibe como parámetros un nombre y una dirección. Compara la dirección con el string que representa cada uno de los posibles movimientos y hace que el personaje con el nombre pasado por parámetro se mueva en la dirección indicada dentro de Minetest.
- **turn_robot:** Función que recibe un nombre y una dirección como parámetros. Compara que la dirección sea válida y hace que el personaje con el nombre indicado como parámetro gire en la dirección indicada dentro del juego.
- **dig_robot:** Recibe un nombre y una dirección esta función. Comprueba que la dirección de la que obtener un material sea una dirección válida y el personaje con el nombre pasado por parámetros intentará obtener un material del nodo indicado en la dirección.
- **place_robot:** Recibe un nombre, el nombre de un material y una dirección. Comprueba que en la dirección indicada se pueda colocar un cubo de material, es decir, que no haya ya un cubo en esa posición. Tras ello el personaje con el nombre indicado por parámetros pondrá en la dirección un cubo del material correspondiente al parámetro recibido. En caso de no disponer de ese material en el inventario no colocará nada.
- **givedrops_robot:** Recibe tres nombres: el del personaje que entregará los materiales, el del receptor y el del material. Obtiene los inventarios de ambos personajes, comprueba que el jugador que va a entregarlos disponga de materiales con el nombre indicado y finalmente los traspasa al inventario del receptor. En caso de que el inventario del emisor no disponga de ese material no entregará nada.

- **add_starter_mat_to_player:** Recibe el nombre de un personaje y el de un material. Obtiene el inventario de dicho jugador y le añade un cubo de material del nombre indicado.
- **clean_inventories:** Recibe el nombre del jugador, el agente y un material. Obtiene el inventario tanto del jugador como del agente y elimina todos los cubos que haya en ellos.
- **spawn_zombiesN:** Recibe el nombre de un zombi y unas coordenadas x, z. Esta función se encarga de añadir al mundo de Minetest un zombi en la parte norte del escenario con el nombre y coordenadas indicadas. También se encarga de girar el modelo del zombi por tal de que empiece la partida mirando a la posición donde se encuentra el agente y el jugador.
- **spawn_zombiesS:** Recibe el nombre de un zombi y unas coordenadas x, z. Esta función se encarga de añadir al mundo de Minetest un zombi en la parte sur del escenario con el nombre y coordenadas indicadas. También se encarga de girar el modelo del zombi por tal de que empiece la partida mirando a la posición donde se encuentra el agente y el jugador.
- **clean_zombies:** Esta función no recibe ningún parámetro. Recorre un array de Lua con llave el nombre del zombi y contenido el objeto del mismo y elimina a todos los zombis del escenario.
- **kill_zombie:** Esta función elimina del mundo de Minetest el zombi cuyo nombre le pasan por parámetros.
- **move_zombie:** Recibe el nombre de un zombi. Primero comprueba que dicho zombi pueda moverse una posición hacia delante. Si es posible avanzará, sino no se moverá.
- **turn_zombie:** Tiene como parámetros el nombre de un zombi y su dirección. Gira al zombi utilizando el nombre y dirección recibidos como parámetros.
- **get_player_pos:** Esta función recibe el nombre de un personaje. Consulta el jugador con este nombre donde se encuentra en el mundo de Minetest y lo devuelve en formato de string.
- **get_pos_in_dir:** Recibe el nombre de un personaje y una dirección. Comprueba las coordenadas que se encuentran en la posición a la que indica la dirección respecto a la posición del personaje y las devuelve en formato de string.
- **get_node_minetest:** Esta función recibe como parámetros coordenadas x, y, z. Observa que cubo se encuentra en la posición marcada por las coordenadas recibidas y devuelve el nombre del material encontrado. En caso de no encontrar nada devolverá “air”.
- **get_player_inventory_item:** Recibe el nombre de un personaje y un número indicando una posición de su inventario. Comprueba si en el inventario del jugador con ese nombre hay algún elemento. En caso afirmativo devolverá el nombre del contenido de esa posición. En caso contrario no devolverá nada.

- **get_player_inventory_item_count:** Recibe el nombre de un personaje y un número indicando una posición de su inventario. Comprueba si en el inventario del jugador con ese nombre hay algún elemento. En caso afirmativo devolverá el número de elementos del contenido de esa posición. En caso contrario devolverá cero. Además, ambos serán strings.
- **get_state_codes:** Recibe el nombre del jugador, el agente, el escenario y una dirección. Comprueba la dirección indicada y mira el contenido que se encuentra en esa dirección respecto a la posición del personaje del agente. Acaba devolviendo un string con el código correspondiente al contenido encontrado.

Algunas de las funciones anteriores no podrían cumplir su función sin la ayuda de las siguientes funciones útiles:

- **gravity:** Se trata de una función que es utilizada tras cualquier acción de un personaje que requiera movimiento. Su utilidad es la de aplicar la gravedad a los jugadores por tal de que no leviten tras realizar un salto sobre un cubo y moverse en cualquier dirección.
- **pos_in_dir:** Función que sin la cual **get_pos_in_dir** no podría hacer su trabajo. Recibe el objeto de un jugador y una dirección. Comprueba en qué dirección está mirando el personaje y una vez ya establecida la orientación mira la posición que se encuentra en la dirección pasada por parámetros.
- **check_inside_world:** Recibe coordenadas x, y, z. Esta función se encarga de comprobar si las coordenadas que le llegan por parámetros se encuentran dentro mundo de Minetest. Devuelve cierto si se encuentran dentro del rango de las coordenadas que forman el mundo. Devolverá falso en caso contrario.

7.2. Implementación de la Simulación

Los scripts de esta parte de la implementación son externos al videojuego Minetest. El lenguaje de programación que se ha utilizado en este apartado es Python para implementar todo lo que comentaremos a continuación.

7.2.1. Mundo de Minetest

Por tal de poder simular el mundo de Minetest en Python, se ha creado una clase con la definición de todas las variables, estructuras y funciones necesarias para hacer dicha tarea. Se trata de una de las partes más importantes del proyecto ya que en el mundo es donde se almacenará todos los movimientos tanto del jugador o del agente o donde se generarán los escenarios con la información de donde se encuentran los cubos de madera, las paredes del laberinto y entre otros.

La estructura por la cual girarán a su alrededor todas las funciones de la clase se tratará de la llamada ***world*/mundo.** El mundo de Minetest será representado en esta estructura que será una lista y esta estará formada por listas de diccionarios. Tendrá un tamaño de 21x21, por lo que habrá 21 listas representando las filas de casillas de un mundo de Minetest y dentro de cada lista habrá 21 diccionarios representando las columnas del mismo mundo. A la vez, estos diccionarios simularan los nodos del videojuego utilizando como llave la altura correspondiente al nodo y como contenido el string del nombre del material o el cubo que ocupa esa posición.

El funcionamiento de esta estructura es bastante sencillo. Y lo explicaré con unos pequeños ejemplos que a la vez sería la explicación detallada de cómo se realiza el trabajo en algunas de las funciones de la clase.

- **Crear un mundo:** Lo primero que hay que hacer antes de generar los escenarios o añadir los jugadores es añadir el suelo sobre el que el agente y el jugador estarán por tal de empezar a crear el mundo. Para ello con llave 4, ya que es la altura a la que se encuentra el suelo en nuestros escenarios de Minetest, se añadirán a todos los diccionarios de la estructura el string con el nombre de la tierra con césped de Minetest. Así obtendremos un mundo de tamaño 21x21 con su suelo preparado.
- **Mover al jugador o el agente:** Se trata de uno de las acciones más usuales durante la simulación de este proyecto o en un mundo de Minetest real. Imaginémonos que el jugador se encuentra en la posición con coordenadas x = 10, y = 5, z = 10 y se mueve en la dirección de las z a z = 11. Lo que sucederá es: Se cogerá el nombre del personaje del diccionario que se encuentra en la posición [10,10] utilizando la llave y, se introducirá el nombre añadiéndolo al diccionario de la nueva posición utilizando la misma llave. Finalmente, se eliminara la llave y contenido del primer diccionario ya que el jugador ya no se encuentra en esa posición.

Una vez explicado en detalle cómo funciona nuestro mundo, pasmos a comentar las funciones que hacen que toda la interacción con el mismo sea posible:

- **create_world:** Esta función da forma al mundo inicializando la estructura **world** con las 21 listas de 21 diccionarios cada una. Para empezar, también le añade el suelo poniendo tierra con césped.
- **init_world:** Se encarga de añadir el jugador y el agente al mundo, e inicializa las estructuras necesarias y llama a las funciones por tal de generar el escenario de entrenamiento indicado en el mismo mundo.

- **init_minetest_world:** Igual que **init_world**, pero para el caso que estemos inicializando el mundo para utilizarlo dentro de Minetest. Las funciones que llama se encargan de realizar los mismos cambios en la estructura **world** como en el mundo del videojuego.
- **get_square:** Dadas unas coordenadas x, z devuelve el diccionario en cuestión que representa esa posición.
- **set_square:** Dadas unas coordenadas x, z y un diccionario, sobrescribe el diccionario existente en esa posición.
- **spawn_player:** Es la función encargada de añadir al mundo al agente y al jugador y de inicializar su inventario. Además, los añade al diccionario llamado **playerlist,** cuya estructura almacenara el objeto de los jugadores con toda su información y utilizando como llave el nombre del jugador.
- **check_inside_world:** Dadas unas coordenadas x, y, z se encarga de validar si dichas coordenadas se encuentran dentro del mundo o no.
- **move_player:** A partir del nombre del jugador o del agente y una dirección realiza la acción de movimiento indicada para el mismo dentro del mundo.
- **turn_player:** Con el nombre y una dirección, el jugador o el agente realizará un giro cambiando su orientación.
- **dig_player:** Utilizando el nombre y una dirección, el jugador o el agente intentarán obtener un cubo de material.
- **place_player:** A partir del nombre del jugador o del agente, una dirección y el nombre de un material, dicho personaje pondrá un cubo de su inventario en esa dirección siempre que esté disponible en el inventario.
- **givedrops_player:** Dando el nombre de un emisor, un receptor y un material, se verificará que el emisor disponga de materiales del tipo indicado y se los entregará al emisor.
- **attack_player:** Función de ataque única para el jugador. Solo puede dar a los zombis 1 golpe y no eliminarlos.
- **attack_ally:** Función de ataque del agente. Esta función permite tanto dañar a los zombis como eliminarlos.
- **attack_ally_minetest:** Función de ataque del agente para el mundo de Minetest. Esta función permite tanto dañar a los zombis como eliminarlos.

También se dispone de funciones útiles para el mundo que explicaremos a continuación:

- **get_pos_in_dir:** Dado el nombre de un jugador dentro del mundo y una dirección, obtiene la posición correspondiente a la dirección indicada teniendo en cuenta su orientación.
- **gravity:** Función que aplica la gravedad después de cualquier movimiento de un jugador por tal de que no leviten.

- **can_advance:** Indica si a la posición que un jugador intenta moverse es posible realizar la acción o no ya que es posible que haya obstáculos.
- **add_cube_to_player_inventory:** Ayuda a **dig_player** realizando la función de añadir un cubo específico que se ha obtenido al inventario del jugador.
- **put_cube_on_place_from_player_inventory:** Función que interviene en el funcionamiento de **place_player**. Coge un cubo de material del inventario y lo añade al mundo.
- **line_of_shigt:** A la hora de atacar no solo hay que tener en cuenta el radio de ataque, sino que también hay que ver que un zombi se encuentre en la línea de visión del jugador por tal de que no pueda atacar a una posición a su espalda.

7.2.2. Jugadores

Se trata de otro de los elementos involucrados con el mundo, pero con información propia. Por tal de simular un jugador de Minetest se ha creado una clase por tal de almacenar los datos necesarios en las variables correspondientes. Tanto el jugador, como el agente y los zombis serán añadidos al mundo como jugadores y sus características serán las siguientes:

- **Nombre/*Name:*** String con el que se identificará cada jugador dentro del mundo.
- **Posición x:** Coordenada x de la posición del personaje.
- **Posición y:** Coordenada y de la posición del personaje.
- **Posición z:** Coordenada z de la posición del personaje.
- **Orientación:** Indica hacia qué punto cardinal está alineada la línea de visión de un personaje (Norte, Sur, Este, Oeste).
- **Inventario:** Será un diccionario donde los jugadores podrán almacenar materiales. La llave será el string del nombre del material y el contenido el número de materiales que han obtenido.
- **Acciones:** Lista con todas las acciones disponibles en el mundo en formato de string.
- **Puntos de vida/HP:** Puntos de salud de los cuales disponen los jugadores antes de ser eliminados.

Las funciones correspondientes a esta clase que ayudan a la gestión de las características de los jugadores y entre otras serían las siguientes:

- **get_pos:** Obtiene la posición del jugador.
- **set_pos:** La posición del jugador pasa a ser aquella que se le pasa por parámetros con coordenadas x, y, z.
- **get_orientation:** Se obtiene la orientación del jugador.

- **set_orientation:** La orientación del jugador es modificada por aquella que se le indica por parámetros.
- **get_inventory:** Devuelve el inventario del jugador correspondiente.
- **set_inventory:** Establece como inventario el pasado por parámetros.
- **get_hp:** Se obtiene los puntos de salud del jugador.
- **set_hp:** Modifica los puntos de salud del jugador.
- **player_act:** Esta función esta implementado todo el comportamiento de nuestro jugador, no del agente, y cambia según el escenario. Es la encargada de hacer cada movimiento del jugador durante los entrenamientos.
- **player_minetest_act:** Igual que **player_act**, pero siendo su versión Minetest ya que hace que sus movimientos muevan su personaje dentro del videojuego.
- **create_players:** Crea al jugador y al agente a partir de su nombre y se le añade las coordenadas de *spawn* que variaran dependiendo el escenario en el que se desarrolla el entrenamiento.
- **create_zombies:** Tiene la misma función que **create_players**, pero con los zombis para el escenario *Fight Zombies.*
- **zombies_act:** Función encargada de los movimientos de los zombis durante el entrenamiento. También está programado su comportamiento.
- **zombies_act_minetest:** Igual que **zombies_act**, pero para Minetest.

Además, por tal de ayudar al jugador en los escenarios *Chase the Player in the Labyrithn* y *Survive the Deadly Road* se ha implementado el algoritmo **dijkstra** para moverse eficientemente desde su posición inicial a una de las posiciones objetivo.

Dijkstra, o algoritmo de caminos mínimos, es un algoritmo que dado un grafo y sus aristas con pesos te da como resultado el camino mínimo entre los dos vértices que se le indiquen. De esta forma en ambos escenarios, si todas las posiciones son vértices del grafo y aquellas que son accesibles entre ellas están conectadas por una arista con peso 1, será fácil para el jugador saber el camino óptimo gracias a este algoritmo.

7.2.3. Escenarios de entrenamiento

En este apartado veremos la inicialización, la generación y características de cada escenario de entrenamiento:

Harvest Wood

El objetivo es la recolección de madera por lo que tendremos que disponer de la información de donde se encuentran y de generarlas dentro del mundo. Este escenario utiliza dos estructuras aparte del **world**: los diccionarios **matspos** y **totalmats**.

- **matspos:** Se trata de un diccionario con llave el string del nombre del material y con contenido una lista de listas de 2 elementos que indican las coordenadas x, z de donde se encuentran las pilas de madera en el mundo. El jugador utiliza esta información por tal de acercarse a las pilas de madera durante el entrenamiento.
- **totalmats:** Es un diccionario con llave el nombre de un material y como contenido el número de cubos del mismo material que hay en el escenario. Se utiliza para contabilizar el total de materiales de cada tipo que hay en un mundo.

En este proyecto y en este escenario se han utilizado solo cubos de madera, pero las estructuras están diseñadas para un trabajo futuro que se le pueda añadir más tipos de materiales para recolectar.

La función que se utiliza para generar los cubos de madera en el mundo se llama **generate_harvest_training**. Funciona de la siguiente manera:

Primero, genera una lista de coordenadas x, z donde se situaran las pilas de madera en el mundo. Las coordenadas son aleatorias y con rangos dentro de nuestro mundo, pero se verifica que no de la casualidad de que dos pilas caigan en el mismo sitio. En el caso que esto suceda, esta pila no se añadirá a la lista y se generará otra. Finalmente, utilizará las coordenadas de posiciones de la lista generada para añadir pilas de madera de 2 cubos en el mundo.

Chase the Player in the Labyrinth

El objetivo de este mapa sería salir del laberinto por lo que hay que disponer de la información relacionada tanto de donde se encuentran las casillas de salida como de los muros que forman el laberinto. Para ello este escenario contará con cuatro estructuras: dos diccionarios llamados **chasepos** y **barriers** y tres listas llamadas **allbarriers, targetpos** y **shortpath**.

- **chasepos:** Se trata de un diccionario que contabilizará a las distancias que se encuentra el agente del jugador durante el entrenamiento para observar su comportamiento
- **barriers**: Es un diccionario que tiene como llave el nombre de los diferentes tipos de barrera, cada uno con una forma, del escenario y su contenido es una lista de listas con las coordenadas x, z de las posiciones donde hay dichas barreras.
- **allbarriers**: Lista compuesta por los diccionarios de **barriers**.
- **targetpos:** Se trata de una lista de listas de coordenadas x, z con las posiciones donde se encuentran las casillas de salida del laberinto.

- **shortpath**: Lista de listas de coordenadas x, z obtenidas del algoritmo **dijkstra**. Es el camino más corto entre la posición inicial del jugador y la casilla más cercana de salida. De esta forma el jugador se moverá posición a posición las coordenadas indicadas y saldrá utilizando el camino óptimo.

La función que se utiliza para generar este escenario se llama **generate_chaser_training** y funciona de la siguiente forma:

Primero de todo, genera una barrera inicial que separa a jugador y agente de la parte noreste del mundo donde hay posibilidad que haya casillas de salida por tal de que no sea demasiado fácil alguna de las configuraciones del escenario. A continuación, se decide de forma aleatoria en que extremo del mundo aparecerán las casillas de salida y se añadirán esas posiciones a **targetpos**. Finalmente, se generaran las barreras escogiendo uno de los cuatro tipos disponibles de forma aleatoria y se añadirán al mundo. Además, conforme se vayan escogiendo se añadirán a **barriers.**

Survive the Deadly Road

El objetivo de este mapa es cruzar el mar de lava por lo que tenemos que tener la información de las posiciones que son seguras donde se encuentra el camino y las casillas objetivo que se encontrarán en la otra punta de este camino por tal de saber si se ha completado el escenario. Este escenario utiliza tres estructuras: el diccionario **roads** y las listas **targetpos** y **shortpath**. Estas listas tienen la misma utilidad que en el escenario anterior, así que pasaremos a comentar el uso del diccionario.

- **roads:** El camino está conectado por piezas con formas diferentes. Esta estructura usa como llave el nombre que codifica esas piezas y tiene como contenido las coordenadas x, z donde se encuentra cada uno de los cubos que las forman.

La función que se utiliza para generar este escenario se llama **generate_deadly_training** y funciona de la siguiente manera:

Primero de todo, genera en el mundo el mar de lava que siempre ocupa las mismas posiciones en el mundo y hace lo mismo con las casillas objetivo que básicamente se trataran de aquellas que se encuentran en la orilla tras pasar el mismo mar. Finalmente, genera el camino desde la parte izquierda del mapa, donde se encuentran el jugador y el agente, hasta la parte derecha del mapa eligiendo entre los cuatro tipos de piezas para el camino de forma aleatoria y conectándolas entre ellas para que al menos siempre haya una forma de pasar de una pieza a otra.

Fight Zombies

El objetivo del mapa es eliminar zombis por lo que tendremos que disponer de toda la información que les corresponde en cada momento. Este escenario utiliza una única estructura llamada **mobs.**

La función que se utiliza en este escenario se llama **generate_zombies** y funciona de forma muy sencilla.

Primero, espera a que la función create_zombies de la clase jugador le entregue un diccionario con el nombre de los zombis y toda su información. Finalmente, esta función añadirá los zombis al mundo en sus posiciones correspondientes y añadirá el objeto con la información de cada uno de ellos en la lista de jugadores del mundo. Se ha comentado las características de **mobs** en un apartado anterior, pero aquí servirá para que durante el entrenamiento sepamos cuantos de los zombis siguen vivos.

Escenarios de entrenamiento de Minetest

Igual que se han comentado estos escenarios de entrenamiento para la simulación, se han tenido que implementar también versiones para poder inicializarlos dentro de Minetest. Básicamente hacen la misma función, pero aparte de modificar el mundo de la simulación también lo hacen con el mundo de Minetest.

7.2.4. Agentes

Para este proyecto se ha tenido que implementar dos tipos de agentes: uno para Q-Learning y SARSA y otro para A2C. Esto es debido a que A2C en comparación con los otros dos algoritmos utiliza 2 redes neuronales con configuraciones diferentes.

Empezaremos comentando las características del agente de Q-Learning y Sarsa:

- **Acciones/*Actions:*** Lista con todas las acciones que puede realizar el agente codificadas en strings.
- **Tamaño de acciones/*Action_size*:** Es el número total de acciones que puede realizar el agente.
- **Tamaño del estado/*State_size:*** Como su nombre sugiere, indica el tamaño del estado.
- ***Learning Rate:*** Es la velocidad de aprendizaje que tendrá el agente, se ha explicado con más detalle en el apartado de algoritmos.
- ***Discount_factor:*** Se trata del factor de descuento y también esta introducido junto al *learning rate* en el apartado de algoritmos.

- **Epsilon:** Se trata del factor de exploración del agente y es un valor dentro del intervalo entre 0 y 1. Indica la frecuencia en la que el agente decidirá explorar en vez de escoger la mejor acción basándose en su experiencia.
- **Modelo/*Model*:** Se trata de la variable donde se construirá la red neuronal.

Para el correcto funcionamiento del agente se utilizan las siguientes funciones:

- **build_model:** Esta función se encarga de construir la red neuronal.
- **get_state:** Teniendo en cuenta la posición del agente y la configuración del estado, obtiene la información del mundo y devuelve el estado correspondiente.
- **get_action:** Función que se encarga de indicarle al agente que acción debe hacer. Aquí es donde se utilizará el parámetro épsilon comentado antes.
- **get_action_minetest:** Mientras que le **get_action** se utiliza para la simulación y dispone de la posibilidad de hacer acciones por tal de explorar, esta función elige acciones una vez se está testeando el agente dentro de Minetest y siempre obtiene la mejor acción correspondiente.
- **get_reward:** Función que a partir de una acción dada la realiza y devuelve la recompensa correspondiente a la misma
- **agent_act:** Se trata de la función que una vez elegida una acción dentro del mundo de Minetest se encarga de realizarla.

El agente de A2C dispone de las mismas características que el anterior y la mayoría de funciones, pero cambia ligeramente lo que comentaremos a continuación:

- **Actor_learning_rate:** Se trata de la velocidad de aprendizaje del *Actor*.
- **Critic_learning_rate:** Es la velocidad de aprendizaje del *Critic*.
- **Actor:** Variable donde se encontrará la red neuronal del *Actor*.
- **Critic:** Variable donde se almacenará la red neuronal del *Critic*.

Viendo las características que han cambiado no habrá mucha sorpresa en las dos funciones de a continuación:

- **build_actor:** Construye la red neuronal del *Actor*.
- **build_critic:** Construye la red neuronal del *Critic*.

7.2.5. Red neuronal

Ha podido ser posible utilizar redes neuronales en este proyecto gracias a su implementación por parte de las librerías de Python **keras** [16] y **tensorflow** [17]. En la documentación de la página oficial de keras se puede aprender fácilmente como construir una red neuronal y sus capas, mientras que tensorflow simplemente lo instalé en el ordenador y no se necesita nada más. Utilicé la versión de tensorflow para GPU.

7.2.6. Algoritmos de aprendizaje por refuerzo

Por tal de implementar los algoritmos de aprendizaje por refuerzo se ha buscado información en libros, páginas web y se ha preguntado al director para entenderlos. A continuación, a partir de esa teoría, repositorios de github que utilizaban diversos algoritmos de aprendizaje por refuerzo en videojuegos de la consola Atari, y páginas web que utilizaban ejemplos con código para enseñar los algoritmos ha sido posible implementarlos y adaptarlos a nuestros scripts. [18] [19] [20]

8. Agentes Q-Learning

El algoritmo de Q-Learning fue el primero en ser implementado y en el que se empezaron a realizar los entrenamientos en los distintos escenarios. Es decir, al ver los resultados de los entrenamientos de estos agentes es cuando se realizaron los cambios pertinentes con la intención de mejorar su comportamiento y después se utilizarían tanto para los algoritmos de SARSA y A2C.

Al principio, se utilizó para el entrenamiento un ratio de aprendizaje de 0.01 y una red neuronal con una entrada de 58 elementos, correspondientes al estado, seguida por capas de 64 y 32 neuronas para acabar con un output de 17 neuronas correspondientes a cada una de las acciones que el agente tiene a su disponibilidad. Dado que la entrada y salida siempre será la misma, siempre se hablara de las otras capas cuando se comenten las modificaciones.

Utilizando este ratio de aprendizaje y esta red neuronal los resultados no eran demasiado buenos y al probar los agentes dentro del Minetest no mostraban ningún tipo de comportamiento, así que se decidió realizar pruebas con ratios de aprendizaje de 0.001 y 0.0001 y probar diferentes configuraciones de la red neuronal. Ya no solo se continuó probando con la configuración de capas con 64 y 32 neuronas sino que se probó también con la configuración de 58 y 29, además de la de 58 y 58.

Estás últimas configuraciones ya dieron resultados y serán las que se muestren en los resultados.

Al utilizar una red neuronal estos agentes utilizarían Deep Q-Learning para realizar su entrenamiento y utilizarán las siguientes características:

- **Factor de exploración = 0.1**
  1 de cada 10 acciones las tomará de forma aleatoria sin tener en cuenta la experiencia del agente por tal de explorar.
- **Factor de descuento = 0.95**
  Se trata de un valor elevado para darle más importancia a la recompensa a largo plazo.
- **Ratio de aprendizaje = 0.001 y 0.0001**

8.1. Harvest in the Wood

8.1.1. Entrenamiento

Sobre las recompensas se empezó con valores pequeños: 10 puntos por recoger un cubo de madera y 100 por recogerlos todos. Viendo que no se veían mucho las diferencias con los otros movimientos se cambió a 50 puntos por madera y 500 al completar el escenario.

Utilizando estos parámetros de recompensas empezó a surgir uno de los comportamientos interesantes que han desarrollado los agentes en este escenario: la explotación de puntos al coger y dejar el mismo cubo de madera. Al principio, el agente obtenía 2 puntos por dejar un cubo en un lugar estratégico que le facilitara el acceso a otro cubo de madera. Entonces aprendió que cogiendo y dejando el mismo cubo obtenía más puntos que no buscando el resto. A partir de entonces la puntuación por dejar cubos pasó a ser 0.

Después de estas experiencias las recompensas pasan a ser las que se encuentran explicadas en el apartado de este escenario. De esta forma la acumulación de materiales pasaba a ser un factor importante y obtendría más punto contra más cubos de madera acumulaba. Así se esperaba que se solucionara la explotación de puntos con el mismo cubo. Además, se decidió añadir recompensas negativas por tal de evitar acciones ilegales como moverse contra una pared o intentar salirse del mapa,

Este escenario también ha sufrido cambios. En los comienzos las pilas de madera estaban formadas por 3 cubos, sim embargo se detectó que el agente nunca era capaz los cubos que se encontraban en la parte superior de la pila. Para intentar simplificar las cosas se redujeron a un tamaño de 2 cubos ya que el objetivo del agente no variaría con esta modificación.

El comportamiento del jugador ha cambiado desde el principio del entrenamiento. Comenzó realizando movimientos en una dirección aleatoria y cogía cubos en caso de encontrarlos delante de él. Sin embargo, al querer que el agente le acompañe y que el objetivo del escenario es que recoja todos los cubos de madera, se decidió que el jugador no recogiera madera para dejarle esa tarea al agente, pero que este se acercará a donde se encontraban las pilas para que el agente pudiera seguirle y localizarlas.

8.1.2. Resultados

Para ver los resultados el factor más importante será la puntuación media que obtienen los agentes en los escenarios, pero también se mostrará su puntuación máxima y la trayectoria media entre el jugador y el agente. La trayectoria es un factor que suma las diferencias de distancias que hay entre el jugador y el agente en cada turno. Por lo que contra más baja sea esta variable más cerca habrá estado el agente del jugador durante el entrenamiento.

Los resultados utilizando las diferentes configuraciones de red neuronal y los diferentes ratios de aprendizajes son los siguientes:

Ratio Aprendizaje 0.001				
Nombre	**Tipo Red Neuronal**	**Puntuación Máxima**	**Puntuación Media**	**Trayectoria Media**
Harvest_27	64/32	515013	59977	6900
Harvest_28	58/29	356263	76198	6830
Harvest_29	58/58	398518	58414	7106

Tabla 13: Resultados Q-Learning Harvest Wood lr=0.001

Ante los resultados obtenidos con este ratio de aprendizaje observamos que **Harvest_28** obtiene una puntuación media de 76198 y es superior a las otras dos opciones. La distancia con el jugador con la trayectoria media es también la inferior, pero no hay tanta diferencia. Viendo que este ha sido el mejor agente, lo observaremos con más detalle:

- **Máximo de cubos de madera recogidos:** 8.
  Habiendo recogido 7 cubos el agente y este se los ha entregado al jugador. Hay que recordar que el jugador empieza con un cubo de madera en el inventario y que no contabiliza para la puntuación del agente.
- **Trayectoria:** 5228.

Ratio Aprendizaje 0.0001				
Nombre	**Tipo Red Neuronal**	**Puntuación Máxima**	**Puntuación Media**	**Trayectoria Media**
Harvest_30	64/32	374271	63313	6819
Harvest_31	58/29	480238	103646	6387
Harvest_32	58/58	454263	87305	6493

Tabla 14: Resultados Q-Learning Harvest Wood lr=0.0001

Los resultados con el ratio de aprendizaje 0.0001 podemos observar que el agente con mejor puntuación media es **Harvest_31** con una puntuación de 103646 seguido por **Harvest_32** que ha obtenido 87305. **Harvest_31** también dispone de la trayectoria media más baja. Observemos en más detalle los resultados de este agente:

- **Máximo de cubos de madera recogidos:** 8.
  El agente ha recogido 7 cubos de madera y se los ha entregado al jugador.
- **Trayectoria:** 5538.

Podemos observar como los resultados obtenidos por los agentes que utilizaron un ratio de aprendizaje de 0.0001 han obtenido mejores puntuaciones que los que han usado el de 0.001. Así que el agente **Harvest_31** será el mejor del algoritmo Q-learning en este escenario.

También podemos ver que las puntuaciones son realmente altas si tenemos en cuenta que completar el escenario recogiendo todos los cubos de madera ofrece una recompensa de 100000 puntos. Esto es debido a que los agentes han aprendido que es mucho más fácil coger y dejar uno de los cubos de madera de su inventario que intentar buscar y recoger otro a esas alturas de la partida.

El objetivo de recoger todos los cubos de madera del escenario no se ha cumplido completamente, pero podemos decir que se ha cumplido el objetivo parcialmente ya que ha recogido cubos y después se los entrega al jugador.

8.2. Chase the Player in the Labyrinth

8.2.1. Entrenamiento

Las recompensas empezaron dando 50 puntos por estar a distancia 1 o 2, 25 puntos a distancia 3 o 4, y 5 puntos si giraba cuando se encontraba una pared delante suyo para esquivarla.

Debido a que los movimientos a distancias cerca del jugador ofrecían una puntuación baja el agente aprendió a ponerse delante de una pared e irla esquivando al girar para obtener una gran puntuación. Para intentar solucionar esto, se aumentaron las recompensas al seguir el jugador y acabaron tal y como están explicadas en el apartado de este escenario. Además, se añadieron recompensas negativas por tal de que no intentara avanzar contra paredes o salir fuera del mapa.

El comportamiento del jugador también ha cambiado desde los inicios. Al principio del entrenamiento su comportamiento era parecido al que se intentó en el escenario de *Harvest*: realizaba movimientos en direcciones aleatorias y en caso de que detectara a su alrededor una casilla de salida se movería hacia ella. De esta forma exploraría todo el escenario. Sin embargo, se decidió cambiar al comportamiento actual utilizando el algoritmo dijkstra por tal de que se moviera de la forma más rápida hacía las casillas de salida y el agente le siguiera.

8.2.2. Resultados

Los resultados obtenidos por los agentes en este escenario son los siguientes:

Ratio Aprendizaje 0.001				
Nombre	**Tipo Red Neuronal**	**Puntuación Máxima**	**Puntuación Media**	**Trayectoria Media**
Chaser_13	64/32	218404	42584	5904
Chaser_14	58/29	203319	9670	7568
Chaser_15	58/58	224951	36104	6567

Tabla 15: Resultados Q-Learning Chase the Player in the Labyrinth lr=0.001

Podemos observar en la tabla 15 que el mejor agente se trata de **Chaser_13** con su puntuación media de 42584 y su trayectoria media es inferior a la de los otros dos. Entre los episodios que el agente ha completado el escenario la máxima puntuación que ha obtenido es de 115720 y sus detalles son los siguientes:

- **Distancias con el jugador:**
  - **1:** 5 veces.
  - **2:** 10 veces.
  - **3:** 8 veces.
  - **4:** 6 veces.
- **Turnos:** 421.
- **Trayectoria:** 3728.

Ratio Aprendizaje 0.0001				
Nombre	**Tipo Red Neuronal**	**Puntuación Máxima**	**Puntuación Media**	**Trayectoria Media**
Chaser_16	64/32	232143	68254	6019
Chaser_17	58/29	29284	29284	7200
Chaser_18	58/58	268305	54301	5878

Tabla 16: Resultados Q-Learning Chase the Player in the Labyrinth lr=0.0001

En la tabla 16 el agente **Chaser_16** obtiene la mejor puntuación media con 68254 puntos, seguido de **Chaser_18** con 54301. El agente restante obtiene una puntuación bastante inferior. Entre los episodios que el agente **Chaser_16** ha completado el escenario la máxima puntuación que ha obtenido es de 116462 y sus detalles son los siguientes:

- **Distancias con el jugador:**
  - **1:** 8 veces.
  - **2:** 9 veces.
  - **3:** 4 veces.
  - **4:** 5 veces.
- **Turnos:** 218.
- **Trayectoria:** 1897.

De nuevo, los agentes con un aprendizaje por refuerzo de 0.0001 han obtenido una mejor puntuación que los de 0.001. Además, podemos observar entre nuestros dos mejores candidatos en sus detalles que **Chaser_16** consigue salir del laberinto en menos turnos y con una trayectoria inferior y, además, encontrándose más usualmente cerca del jugador a distancia 1 y 2 que **Chaser_13**. De esta forma **Chaser_16** se convierte en el mejor agente de Q-learning para este escenario.

También podemos ver en estos resultados que las puntuaciones máximas son realmente altas si las comparamos con las que se han obtenido al completar el escenario. Esto es debido a episodios que el agente ha seguido al jugador como debería, pero no ha llegado a encontrar una de las casillas de salida, así que se ha seguido sumando la puntuación al irse moviendo a una distancia cercana del jugador.

El objetivo de este escenario se ha cumplido ya que los agentes han conseguido salir del laberinto.

8.3. Survive the Deadly Road

8.3.1. Entrenamiento

La puntuación empezó también siendo baja como en los otros escenarios. El moverse por el camino a una distancia de 5 posiciones o inferior se recompensaba con 50 puntos y si era una superior obtenía 10 puntos, el mismo número que al realizar un giro por tal de evitar la lava. Al ver como evolucionaban los dos escenarios anteriores se aumentó las recompensas para seguir al jugador de 50 puntos a 1000 por encontrarse a una

distancia de 5 o menos y de 10 a 100 para las distancias mayores a 5. Además se introdujo una gran penalización de -100 puntos en caso de que cayera a la lava.

Con esta configuración de recompensas, los agentes se dieron cuenta que ir por el camino y caer a la lava solo harían que perder puntos, por lo que se quedaban en la orilla del principio, avanzaban a un lugar que tuvieran lava en frente y empezaban a realizar giros por ganar esos 10 puntos. De esta forma se mantenían vivos y conseguían más puntos que intentando cruzar por el camino a la otra orilla y conseguir cruzar el mar de lava.

Para intentar solucionar este comportamiento se redujo la recompensa de evitar la lava a 1 punto.

Sobre el comportamiento del agente ya se había implementado el dijkstra para el escenario *Chase* así que ya se utilizó directamente en este escenario el seguimiento del camino óptimo.

8.3.2. Resultados

Ratio Aprendizaje 0.001				
Nombre	**Tipo Red Neuronal**	**Puntuación Máxima**	**Puntuación Media**	**Trayectoria Media**
Deadly_19	64/32	6300	1767	67
Deadly_20	58/29	6200	3067	82
Deadly_21	58/58	5100	1017	137

Tabla 17: Resultados Q-Learning Survive the Deadly Road lr=0.001

Podemos observar como el agente **Deadly_20** obtiene una puntuación superior al resto con 3067 puntos de media en la tabla 17. Veamos los detalles del episodio donde ha conseguido su puntuación máxima:

- **Turnos sobrevividos:** 13.
- **Trayectoria:** 80

Ratio Aprendizaje 0.0001				
Nombre	**Tipo Red Neuronal**	**Puntuación Máxima**	**Puntuación Media**	**Trayectoria Media**
Deadly_22	64/32	6501	2392	339
Deadly_23	58/29	5900	2179	80
Deadly_24	58/58	4400	1275	85

Tabla 18: Resultados Q-Learning Survive the Deadly Road lr=0.0001

En la tabla 18 podemos ver como los agentes **Deadly_22** y **Deadly_23** tienen una puntuación media muy similar. En cambio, si comparamos las puntuaciones máximas el primero obtiene una diferencia de 600 puntos sobre el segundo. Veamos los detalles del episodio donde **Deadly_22** ha conseguido su puntuación máxima:

- **Turnos sobrevividos:** 272.
- **Trayectoria:** 6181.

El agente **Deadly_30** obtiene una mayor puntuación media que **Deadly_22**. Teniendo eso en cuenta y la trayectoria media podemos observar como el primer agente intenta seguir al jugador por el camino, pero acaba eliminado. En cambio, el segundo agente se queda a más distancia por tal de estar más seguro. En conclusión, **Deadly_30** se convierte en el mejor agente de Q-learning en este escenario.

El objetivo de este escenario no se ha cumplido ya que ninguno de los agentes ha conseguido cruzar el camino a la otra orilla.

8.4. Fight Zombies

8.4.1. Entrenamiento

Este escenario ha sido el último en realizar los entrenamientos así que se han ido aplicando los conocimientos obtenidos de los otros tres escenarios. Las recompensas para este escenario fueron desde el principio las que se indican en el apartado de este escenario.

Hay que tener en cuenta a la hora de analizar los resultados que cada vez que el jugador es atacado por un zombi el agente obtiene una penalización de -100 puntos por tal de que intente eliminar a los zombis de la forma más eficiente posible.

El jugador utiliza la misma estrategia que en el escenario de *Harvest* y sabe de la posición de todos los zombis del mundo para actuar de acuerdo a ello.

8.4.2. Resultados

Ratio Aprendizaje 0.001				
Nombre	**Tipo Red Neuronal**	**Puntuación Máxima**	**Puntuación Media**	**Trayectoria Media**
Fight_4	64/32	6584	-1317	5886
Fight_5	58/29	6899	-457	6403
Fight_6	58/58	7836	-935	5907

Tabla 19: Resultados Q-Learning Fight Zombies Road lr=0.001

Podemos observar en la tabla 19 que todas las puntuaciones medias son negativas. Esto es debido a que entre todos los episodios no en todo consigue eliminar a suficientes zombis como para que la media sea positiva. En este caso, el mejor agente será aquel que se encuentre con una puntuación media más cercana a ser positiva. Claramente en esta apartado tenemos al agente **Fight_5** y estos son los detalles del episodio que ha conseguido su puntuación máxima:

- **Zombis eliminados:** 4.
- **Turnos:** 29.
- **Trayectoria:** 358.

Ratio Aprendizaje 0.0001				
Nombre	**Tipo Red Neuronal**	**Puntuación Máxima**	**Puntuación Media**	**Trayectoria Media**
Fight_7	64/32	6807	-789	6412
Fight_8	58/29	7023	-630	6693
Fight_9	58/58	7552	-993	6635

Tabla 20: Resultados Q-Learning Fight Zombies Road lr=0.0001

En la tabla 20 no hay una gran diferencia en la puntuación de los 2 primeros agentes, mientras que el tercero ya se distancia un poco. Teniendo en cuenta que lo que queremos es el agente que obtenga una puntuación más cercana a 0, en este caso ganaría **Fight_8**. Sus detalles en el episodio con su puntuación máxima son los siguientes:

- **Zombis eliminados:** 4.
- **Turnos:** 246.
- **Trayectoria:** 5182.

Entre los mejores de los ratios 0.001 y 0.0001 el ganador de Q-learning para el escenario será **Fight_5**. Como podemos ver en los detalles de cada uno este agente elimina a los zombis en un número bastante inferior de turnos que **Fight_9**. Traduciendo esta diferencia en que son turnos que los zombis no dañaran al jugador.

El agente ha aprendido a quedarse en el centro del escenario y atacar aquellos zombis que se acercan ya que ellos tienen que acercarse a la parte central por tal de encontrar al jugador.

El objetivo de este escenario se ha cumplido ya que los agentes han conseguido eliminar a los 4 zombis.

9. Agentes SARSA

Los agentes de SARSA han seguido las mismas configuraciones de red neuronal y *learning rate* que los de Q-learning. Al utilizar una red neuronal estos agentes utilizarían Deep SARSA para realizar su entrenamiento y utilizarán las siguientes características:

- **Factor de exploración = 0.1**
- **Factor de descuento = 0.95**
- **Ratio de aprendizaje = 0.001 y 0.0001**

Dado que en el apartado de Q-learning ya se han especificado los detalles de entrenamiento de cada escenario, no aparecerá ese subapartado para estos agentes ya que se repetiría lo mismo.

9.1. Harvest in the Wood

9.1.1. Resultados

Ratio Aprendizaje 0.001				
Nombre	**Tipo Red Neuronal**	**Puntuación Máxima**	**Puntuación Media**	**Trayectoria Media**
Harvest_4	64/32	199154	4620	5788
Harvest_5	58/29	5995	96	5595
Harvest_6	58/58	252013	5606	6091

Tabla 21: Resultados SARSA Harvest Wood lr=0.001

Si observamos la tabla 21 podemos ver que el mejor agente ha sido **Harvest_6** con una puntuación media de 5606. Los detalles de este agente son los siguientes:

- **Máximo de cubos de madera recogidos:** 11.
  El agente ha recogido 10 cubos y ha acabado entregando 8 al jugador de forma que el agente ha acabado la partida al recoger todos los cubos y se ha quedado con 2 en el inventario.
- **Turnos:** 379.
- **Trayectoria:** 4806.

Ratio Aprendizaje 0.0001				
Nombre	**Tipo Red Neuronal**	**Puntuación Máxima**	**Puntuación Media**	**Trayectoria Media**
Harvest_7	64/32	463495	102708	6563
Harvest_8	58/29	374004	79659	7258
Harvest_9	58/58	699042	87994	6932

Tabla 22: Resultados SARSA Harvest Wood lr=0.0001

En la tabla 22 podemos observar como **Harvest_7** obtiene una puntuación media de 102708, mientras que en segundo lugar **Harvest_9** obtiene 87994. Si se eligiera en este caso en base a la puntuación ganaría el primer agente, sin embargo, el segundo agente ha llegado completar el escenario. Así que elegiremos a **Harvest_9** como el mejor del ratio 0.0001 y veremos sus detalles:

- **Máximo de cubos de madera recogidos:** 11.
  El agente ha recogido 10 cubos y ha acabado entregando 8 al jugador de forma que el agente ha acabado la partida al recoger todos los cubos y se ha quedado con 2 en el inventario.
- **Turnos:** 457.
- **Trayectoria:** 3861.

La decisión del mejor agente para este escenario será un poco diferente en este caso. En vez de comparar la puntuación media se tendrá en cuenta el episodio en que han conseguido recoger todos los cubos de madera del escenario. Hasta este punto ninguno de los agentes había conseguido completarlo y estos lo han conseguido uno cada uno. Dado que ambos cumplen con el objetivo el mejor agente de SARSA para este escenario se tratará de **Harvest_9**. Aunque tarde más en cumplir este objetivo su trayectoria es de 1060 menos, lo que significa que ha estado acompañando al jugador mucho más.

Se puede observar como los agentes del ratio 0.0001 tienen unas puntuaciones medias y máximas desorbitadas en comparación a los del ratio 0.001. Esto es debido a que en episodios estos agentes han aprendido a utilizar la estrategia de coger y dejar el cubo para conseguir más puntos cuando ya tenían bastantes puntos acumulados en su inventario.

El objetivo de este escenario se ha conseguido ya que dos agentes han llegado a recolectar todos los cubos de madera del escenario.

9.2. Chase the Player in the Labyrinth

9.2.1. Resultados

Ratio Aprendizaje 0.001				
Nombre	**Tipo Red Neuronal**	**Puntuación Máxima**	**Puntuación Media**	**Trayectoria Media**
Chaser_8	64/32	205210	22498	7940
Chaser_9	58/29	221894	16966	7204
Chaser_10	58/58	223941	22261	6954

Tabla 23: Resultados SARSA Chase the Player in the Labyrinth lr=0.001

En la tabla 23 podemos ver como los agentes **Chaser_8** y **Chaser_10** disponen de las mejores puntuaciones medias, pero son muy parecidas. Si pasamos a tener en cuenta otra métrica como lo es la trayectoria que indica lo cerca que se ha encontrado durante el entrenamiento el agente del jugador entonces ganaría el segundo ya que es mejor por una diferencia de unos 1000. Entre los episodios que **Chaser_10** ha completado el escenario su puntuación máxima ha sido de 196039 y sus detalles son los siguientes:

- **Distancias con el jugador:**
  - **1:** 40 veces.
  - **2:** 57 veces.
  - **3:** 1 veces.
  - **4:** 0 veces.
- **Turnos:** 142.
- **Trayectoria:** 203.

Ratio Aprendizaje 0.0001				
Nombre	**Tipo Red Neuronal**	**Puntuación Máxima**	**Puntuación Media**	**Trayectoria Media**
Chaser_11	64/32	226005	54318	5597
Chaser_12	58/29	233240	68657	5623
Chaser_13	58/58	463651	33666	7147

Tabla 24: Resultados SARSA Chase the Player in the Labyrinth lr=0.0001

Podemos observar en la tabla 24 como el agente con una mayor puntuación media se trata de **Chaser_12**. Entre los episodios en que ha completado este escenario su puntuación máxima es de 107006 y sus detalles son los siguientes:

- **Distancias con el jugador:**
  - **1:** 5 veces.
  - **2:** 3 veces.
  - **3:** 1 veces.
  - **4:** 2 veces.
- **Turnos:** 287.
- **Trayectoria:** 4339.

Teniendo en cuenta los dos agentes elegidos el mejor se trataría de **Chaser_10**. No solo se encuentra muchas más veces a una distancia de 1 o 2 posiciones del jugador que el otro agente sino que termina en un número inferior de turnos y con una trayectoria mucho más inferior que la de **Chaser_12**.

El objetivo de este escenario se ha cumplido ya que los agentes han conseguido salir del laberinto satisfactoriamente.

9.3. Survive the Deadly Road

9.3.1. Resultados

Ratio Aprendizaje 0.001				
Nombre	**Tipo Red Neuronal**	**Puntuación Máxima**	**Puntuación Media**	**Trayectoria Media**
Deadly_14	64/32	6100	1389	113
Deadly_15	58/29	6101	1904	1639
Deadly_16	58/58	4404	1604	1186

Tabla 25: Resultados SARSA Survive the Deadly Road lr=0.001

Podemos ver en la tabla 25 que no hay grandes diferencias entre los tres agentes, sin embargo, el que dispone de una puntuación media superando en 300 puntos al segundo clasificado se trata de **Deadly_15**. Veamos los detalles del episodio donde ha obtenido su puntuación máxima:

- **Turnos sobrevividos:** 40.
- **Trayectoria:** 764.

Ratio Aprendizaje 0.0001				
Nombre	**Tipo Red Neuronal**	**Puntuación Máxima**	**Puntuación Media**	**Trayectoria Media**
Deadly_17	64/32	4900	1408	154
Deadly_18	58/29	3900	352	106
Deadly_19	58/58	6401	1306	87

Tabla 26: Resultados SARSA Survive the Deadly Road lr=0.0001

Entre los agentes con el ratio de 0.0001 la mejor elección es la del agente **Deadly_17** que sobrepasa al segundo clasificado solo por 100 puntos en la puntuación media. Los detalles del episodio donde ha obtenido su puntuación máxima son los siguientes:

- **Turnos sobrevividos:** 8.
- **Trayectoria:** 32.

Entre los dos candidatos posibles si tuviera que elegir a uno como mejor sería **Deadly_17**. Esta elección es debida a que aunque el agente sobrevive pocos turnos al menos intenta seguir al jugador en comparación con el otro agente.

El objetivo de este escenario no se ha cumplido ya que ninguno de los agentes ha conseguido cruzar el mar de lava satisfactoriamente.

9.4. Fight Zombies

9.4.1. Resultados

Ratio Aprendizaje 0.001				
Nombre	**Tipo Red Neuronal**	**Puntuación Máxima**	**Puntuación Media**	**Trayectoria Media**
Fight_1	64/32	7335	-1357	5710
Fight_2	58/29	7614	-1299	6378
Fight_3	58/58	7258	-807	6395

Tabla 27: Resultados SARSA Fight Zombies Road lr=0.001

Si aplicamos el mismo método que hemos realizado anteriormente con este escenario el ganador de estos agentes sería **Fight_3** siendo el que tiene la puntuación media más cercana a los puntos positivos. Los detalles del episodio con la puntuación máxima es la siguiente:

- **Zombis eliminados:** 4.
- **Turnos:** 334.
- **Trayectoria:** 5478.

Ratio Aprendizaje 0.0001				
Nombre	**Tipo Red Neuronal**	**Puntuación Máxima**	**Puntuación Media**	**Trayectoria Media**
Fight_4	64/32	7844	-943	6717
Fight_5	58/29	6898	-1141	5772
Fight_6	58/58	7241	-1028	6671

Tabla 28: Resultados SARSA Fight Zombies Road lr=0.0001

Entre los agentes que han utilizado el ratio de aprendizaje de 0.0001 no hay ninguno que destaque en especial ya que sus puntuaciones medias son todas muy parecidas. Sin embargo, para mantener el criterio de elección escogeremos a **Fight_4** ya que es el que tiene la puntuación más cercana a 0. Los detalles del episodio con la puntuación máxima es la siguiente:

- **Zombis eliminados:** 4.
- **Turnos:** 108.
- **Trayectoria:** 1766.

Claramente el ganador entre los dos candidatos teniendo en cuenta los turnos en los que elimina a los 4 zombis y su trayectoria se trata del agente **Fight_4**. Al estar más cerca del jugador y acabar en menos turnos evita que reciba más daño así que sería el mejor agente de SARSA para este escenario.

El objetivo de este escenario se ha conseguido ya que los agentes han eliminado a todos los zombis.

10. Agentes A2C

Los agentes de A2C han seguido las mismas configuraciones de red neuronal y *learning rate* tanto para el *Actor* como para el *Critic*. Viendo que ninguna configuración daba ningún resultado, se optó a quitar una de las capas ocultas por tal de mantener estas dos redes neuronales de la forma más simple posible. Pese a todos los esfuerzos, pruebas y configuraciones diferentes no se ha obtenido ningún resultado con este algoritmo.

Esto puede ser a que le costara mucho más aprender que a los algoritmos de Q-learning y SARSA debido a que tienen que aprender ambas redes neuronales en paralelo. Por tal de mostrar resultados se ha utilizado los parámetros similares a los otros dos algoritmos:

- **Factor de exploración = 0.1**
- **Factor de descuento = 0.9**
- **Ratio de aprendizaje Actor = 0.001 y 0.0001**
- **Ratio de aprendizaje Critic = 0.005 y 0.0005**

10.1. Harvest in the Wood

10.1.1. Resultados

Nombre	Ratio de Aprendizaje Actor/Critic	Puntuación Máxima	Puntuación Media	Trayectoria Media
Harvest_11	0.001/0.005	-486	-486	8500
Harvest_12	0.0001/0.0005	0	0	4298

Tabla 29: Resultados A2C Harvest Wood

El objetivo de este escenario no se ha cumplido ya que los agentes no han realizado ningún tipo de comportamiento.

10.2. Chase the Player in the Labyrinth

10.2.1. Resultados

Nombre	Ratio de Aprendizaje Actor/Critic	Puntuación Máxima	Puntuación Media	Trayectoria Media
Chaser_3	0.001/0.005	3	0	5397
Chaser_4	0.0001/0.0005	0	0	5397

Tabla 30: Resultados A2C Chase the Player in the Labyrinth

El objetivo de este escenario no se ha cumplido ya que los agentes no han realizado ningún tipo de comportamiento.

10.3. Survive the Deadly Road

10.3.1. Resultados

Nombre	Ratio de Aprendizaje Actor/Critic	Puntuación Máxima	Puntuación Media	Trayectoria Media
Deadly_2	0.001/0.005	0	0	10975
Deadly_3	0.0001/0.0005	0	0	11345

Tabla 31: Resultados A2C Survive the Deadly Road

El objetivo de este escenario no se ha cumplido ya que los agentes no han realizado ningún tipo de comportamiento.

10.4. Fight Zombies

10.4.1. Resultados

Nombre	Ratio de Aprendizaje Actor/Critic	Puntuación Máxima	Puntuación Media	Trayectoria Media
Fight_1	0.001/0.005	-1342	-1974	5104
Fight_2	0.0001/0.0005	-366	-1873	5090

Tabla 32: Resultados A2C Fight Zombies

El objetivo de este escenario no se ha cumplido ya que los agentes no han realizado ningún tipo de comportamiento.

11. Conclusiones

Hemos logrado establecer una conexión entre nuestros scripts de Python con el videojuego Minetest con el que queríamos trabajar en este proyecto. Con esta conexión, que al final se ha establecido mediante el envío de mensajes desde nuestro código, hemos podido simular nuestros agentes dentro del juego.

El algoritmo de A2C ha resultado no funcionar en este proyecto, por lo que habría que mirar para un trabajo futuro alguna variante parecida para ver si se llegaría a obtener algún resultado en unos escenarios mediante la utilización de *Actor* y *Critic* de forma paralela.

Los algoritmos de Q-learning y SARSA han conseguido obtener buenos resultados en los escenarios de entrenamiento, excepto en el tercer escenario en el que el agente tenía que cruzar un mar de lava utilizando un camino de acero.

El fracaso en este escenario de entrenamiento es debido a que ha resultado demasiado difícil para el agente. Las piezas del camino estaban conectadas las unas con las otras solo en una casilla del final de una con la del principio de la otra. Esto requeriría tener una precisión para avanzar y no caerse a la lava. Una precisión que los agentes no han conseguido obtener.

A continuación sacaremos las conclusiones de los otros 3 escenarios que han conseguido cumplir los objetivos, para ver cuál de los algoritmos ha sido mejor en cada escenario.

Harvest Wood				
Algoritmo	**Nombre**	**Max. Cubos de Madera Recogidos**	**Trayectoria**	**Puntuación Media**
Q-learning	**Harvest_31**	8	5538	103646
SARSA	**Harvest_9**	11	3861	87994

Tabla 33: Conclusiones Harvest Wood

En el escenario *Harvest Wood* el algoritmo que ha obtenido el mejor agente se trata de SARSA ya que es el único que ha conseguido recolectar todos los cubos de madera y cumplir el objetivo.

Chase the Player in the Labyrinth								
Algoritmo	**Nombre**	**Dist1**	**Dist2**	**Dist3**	**Dist4**	**Turn.**	**Tray.**	**Puntuación Media**
Q-Learning	**Chaser_16**	8	9	4	5	218	1897	68254
SARSA	**Chaser_10**	40	57	1	0	142	203	22261

Tabla 34: Conclusiones Chase the Player in the Labyrinth

En el escenario *Chase the Player in the Labyrinth* ambos algoritmos han obtenido buenos resultados, pero el mejor agente de ambos lo ha obtenido Q-learning. Puede que pierda comparando sus dos mejores episodios, pero su mayor puntuación media indica que ha completado más veces satisfactoriamente el escenario que el agente de SARSA.

Fight Zombies					
Algoritmo	**Nombre**	**Zombis Eliminados**	**Turnos**	**Trayectoria**	**Puntuación Media**
Q-learning	**Fight_5**	4	29	358	-457
SARSA	**Fight_4**	4	108	1766	-943

Tabla 35 : Conclusiones Fight Zombies

En el escenario *Fight Zombies* ambos algoritmos han obtenido buenos resultados, pero el mejor agente lo ha obtenido Q-learning. Este gana a SARSA tanto comparando su mejor episodio como en puntuación media.

En conclusión a los 3 escenarios, Q-learning ha ganado en 2 de ellos y SARSA en 1.

12. Opinión Personal

Realizar este proyecto ha sido una gran experiencia personal. Ha sido la primera vez que me he enfrentado a un proyecto de esta magnitud y que además no había un libro de teoría o apuntes definidos para el tema en cuestión como podría suceder en una asignatura de la carrera. Se ha tenido que investigar tanto a través de internet como en libros y tomar decisiones para realizar tareas que después se compartirían y analizarían con el director de proyecto. Una forma de trabajar que no había experimentado nunca y que me ha permitido ir avanzando durante el proyecto.

Para la elección de este proyecto yo ya estaba interesado en la inteligencia artificial gracias a la asignatura de IA y, además, soy un gran fan de los videojuegos. Así que este proyecto ha sido la mejor forma de acabar los años de carrera en la FIB para mí. Ya no solo trabajando con IA y videojuegos, sino que profundizando en la inteligencia artificial y descubriendo unos de sus campos que no conocía, y no se había dado en ninguna asignatura del grado que haya cursado, llamado aprendizaje por refuerzo y me ha encantado. He aprendido mucho de esta rama, empezando desde sus características hasta los 3 algoritmos que se han implementado en el proyecto.

Nunca he sido un gran fan de los videojuegos de tipo voxel como Minetest o Minecraft. Mi género favorito para los videojuegos es el de los RPG (*Role Playing Game)* donde tienes unos personajes que viven una aventura con su historia y van subiendo de nivel mientras gestionas sus habilidades y equipamiento. En cambio, Minetest es un juego sin historia ni gestión de habilidades, simplemente haces lo que quieres en el momento que te apetece. Por estas razones trabajar con Minetest ha sido un reto a la hora de diseñar tareas para el entrenamiento de los agentes y sus escenarios.

Además, el entrenamiento de los agentes ha resultado una parte muy divertida e interesante, aunque hay que reconocer que también frustrante de vez en cuando. El hecho de poder ver los puntos que obtenían, que comportamientos iban aprendiendo e incluso de algunos “malos hábitos” que después había que pensar que se podía cambiar en el sistema de recompensas del agente para eliminarlos o que no comprometieran el entrenamiento en un escenario. Esta parte ha sido como un juego donde agente tras agente tenía que ver que configuración de recompensas mejoraban su comportamiento. Aún mejor cuando ves dentro del juego de Minetest las diferencias de cuando el agente no llegaba a moverse o se quedaba estancado en una parte del escenario a cuando realmente realiza pasos del buen comportamiento que queríamos.

Para terminar, estoy muy contento por el resultado del proyecto y orgulloso de todas las horas que le he dedicado. Algunas muy divertidas, pero sobre todo orgulloso por aquellas veces en que las cosas no funcionaban, aparecían *bugs*, o que parecía imposible el poder conectar mis scripts de Python con el juego. Ya que en vez de rendirme he continuado trabajando y dedicándole horas por tal de solucionar los problemas o encontrar alternativas.

13. Trabajo Futuro

Como trabajo futuro se proponen diferentes opciones:

13.1. Probar mejoras de nuestros algoritmos

- **Double Deep Q-Learning:** Este algoritmo es como el Q-learning que utilizamos en este proyecto, pero añadiéndole una segunda red. Esta mejora trata de disminuir la sobreestimación que suele provocar DQN utilizando la segunda red para estimar los Q-values correspondientes a las acciones que toma la primera red.
- **A3C:** Dado el resultado que hemos obtenido de A2C en este proyecto, sería bueno utilizar una mejora de este. A3C funciona igual que A2C, pero sin utilizar *experience replay*. En vez de utilizarlo lanza diferentes threads que serán diferentes agentes por tal de explorar el espacio de estados más rápidamente y explotar el paralelismo del *Actor* y el *Critic*.
- **PPO (Proximal Policy Optimization):** Se trata de un algoritmo *on-policy* que trata de hacer mejoras en su comportamiento lo más grande posibles, pero sin separarse en gran distancia de su comportamiento antiguo por tal de no provocar un colapso en el rendimiento.

13.2. Modificar nuestros escenarios/Añadir escenarios

- **Harvest Wood:** En este escenario el agente tenía que recoger madera, pero se podrían añadir más materiales con diferentes puntuaciones para que el agente tuviera que recogerlos también.
- **Survive the Deadly Road:** Se podrían realizar plataformas con mayores conexiones entre ellas por tal de facilitar el escenario a los agentes.
- **Construcción de edificios:** Uno de los aspectos más entretenidos en los juegos de voxel como Minetest es poder construir desde una casa de madera a un castillo con paredes hechas de oro. Se podría experimentar para saber hasta que complejidad de estructuras podría aprender a construir la IA mediante el aprendizaje por refuerzo.

14. Bibliografía

[1] “Market for Artificial Intelligence Projected to Hit $36 Billion by 2025 | TOP500 Supercomputer Sites.” Accessed September 21, 2018. https://www.top500.org/news/market-for-artificial-intelligence-projected-to-hit-36-billion-by-2025/.

[2] Georgios N. Yannakakis and Julius Togelius. *Artificial Intelligence and Games*. Springer, Cham, 2018.

[3] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second Edition, MIT Press, Cambridge, 2018.

[4] “Minetest - Minetest Wiki.” Accessed September 23, 2018. https://wiki.minetest.net/Minetest.

[5] “Project Malmo.” *Microsoft Research* (blog). Accessed September 24, 2018. https://www.microsoft.com/en-us/research/project/project-malmo/.

[6] “DeepMind and Blizzard Open StarCraft II as an AI Research Environment.” DeepMind. Accessed September 24, 2018. https://deepmind.com/blog/deepmind-and-blizzard-open-starcraft-ii-ai-research-environment/.

[7] DeepMind. *RL Course by David Silver - Lecture 1: Introduction to Reinforcement Learning*. Accessed September 29, 2018. https://www.youtube.com/playlist?list=PLqYmG7hTraZDM-OYHWgPebj2MfCFzFObQ

[8] “Sueldos En Jefe de Proyecto En España | Indeed.Es.” Accessed October 14, 2018. https://www.indeed.es/salaries/Jefe-de-proyecto-Salaries.

[9] “Sueldos En Programador/a Junior En España | Indeed.Es.” Accessed October 14, 2018. https://www.indeed.es/salaries/Programador/a-junior-Salaries.

[10] “Los ordenadores también emiten CO2.” Ecoembes, April 12, 2016, Accessed October 14, 2018. https://www.ecoembes.com/es/planeta-recicla/blog/los-ordenadores-tambien-emiten-co2.

[11] “A Quick Introduction to Neural Networks – the Data Science Blog.” Accessed December 29, 2018. https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/.

[12] “Demystifying Deep Reinforcement Learning | Computational Neuroscience Lab.” Accessed December 29, 2018. https://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/.

[13] Huang, 黃功詳 Steeve. “Introduction to Various Reinforcement Learning Algorithms. Part I (Q-Learning, SARSA, DQN, DDPG).” Towards Data Science, Accessed December 29, 2018. https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287.

[14] Simonini, Thomas. “Diving Deeper into Reinforcement Learning with Q-Learning.” freeCodeCamp.org, April 10, 2018, Accessed December 29, 2018. https://medium.freecodecamp.org/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe.

[15] “An Intro to Advantage Actor Critic Methods: Let’s Play Sonic the Hedgehog!” Accessed December 29, 2018. https://medium.freecodecamp.org/an-intro-to-advantage-actor-critic-methods-lets-play-sonic-the-hedgehog-86d6240171d.

[16] “Home - Keras Documentation.” Accessed October 15, 2018. https://keras.io/.

[17] “TensorFlow.” TensorFlow. Accessed January 13, 2019. https://www.tensorflow.org/?hl=es

[18] *Deep Reinforcement Learning for Keras*. Python. 2016. Reprint, Keras-RL, 2018, Accessed October 15, 2018. https://github.com/keras-rl/keras-rl.

[19] *Minimal and Clean Reinforcement Learning Examples.* Python. 2017. Reprint, RLCode, 2019, Accessed October 15, 2018. https://github.com/rlcode/reinforcement-learning.

[20] “Qmaze.” Accessed October 15, 2018. https://www.samyzaf.com/ML/rl/qmaze.html.